SAMS

计算机技术 详林 精选系列

24小时学通Qt编程

Daniel Solin 著

袁鹏飞 译

人民邮电出版社
www.pptph.com.cn

计算机技术
译林
精选系列

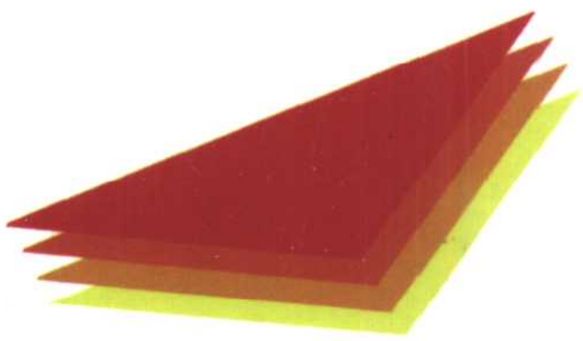
SAMS 计算机技术 译林 精选系列

24小时学通Qt编程

Daniel Solin 著
袁鹏飞 译

人民邮电出版社
www.pptph.com.cn

封面设计 胡平利



Qt 是用于创建 Linux 下的图形程序最出色的工具包之一。它是众所周知的桌面环境 KDE 所使用的工具包。Qt 不仅适用于 UNIX/Linux，而且还可用于 Windows。

本书从程序设计角度全面介绍 Linux/UNIX 下基于 Qt 的图形界面程序开发方法。全书共分六部分：第一部分 Qt 基础知识，介绍面向对象程序设计、槽和信号、Qt 构造块等内容。第二部分重要的 Qt 部件，介绍常用 Qt 部件使用方法、怎样绘制图形和创建对话框等。第三部分深入学习 Qt，介绍布局管理器、文件和目录、文本和常规表达式、容器类、图形、程序通信等内容。第四部分 Qt 编程技巧，介绍怎样编写 KDE 应用程序、使用 OpenGL 类绘图和创建 Netscape 插件等。第五部分改善程序性能，介绍 Qt 程序的国际化、移植、调试等问题，以及怎样使用构造程序简单快捷地创建图形界面。

全书循序渐进，难度适中，实用性和操作性强，适用于 Linux/UNIX 和 Windows 下的 GUI 程序开发人员学习使用。

读者指南

入门 初级 中级 专家

ISBN 7-115-08849-7

9 787115 088499 >

ISBN7-115-08849-7/TP·1869
定价:37.00 元

人民邮电出版社
www.pptph.com.cn

计算机技术译林精选系列

24小时学通Qt编程

Daniel Solin 著

袁鹏飞 译

人民邮电出版社

图书在版编目（CIP）数据

24 小时学通 Qt 编程/（美）索林（Solin,P.）著；袁鹏飞译。—北京：人民邮电出版社，2000.10

（计算机技术译林精选系列）

ISBN 7-115-08849-7

Ⅰ. 2… Ⅱ. ①索… ②袁… Ⅲ. 软件工具，QT—程序设计

Ⅳ. TP311.56

中国版本图书馆 CIP 数据核字（2000）第 75297 号

计算机技术译林精选系列

24 小时学通 Qt 编程

◆ 著　　Daniel Solin

译　　袁鹏飞

责任编辑　刘　涛

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号

邮编　100061　　电子函件　315@ pptph.com.cn

网址　http://www.pptph.com.cn

北京汉魂图文设计有限公司制作

北京鸿佳印刷厂印刷

新华书店总店北京发行所经销

◆ 开本：787 × 1092　1/16

印张：22

字数：526 千字　　2000 年 11 月第 1 版

印数：1 - 5 000 册　　2000 年 11 月北京第 1 次印刷

著作权合同登记　图字：01 - 2000 - 2857 号

ISBN 7-115-08849-7/TP·1869

定价：37.00 元

版权声明

Daniel Solin: SAMS Teach Youself Qt™ Programming in 24 Hours

Authorized translation from the English language edition published by SAMS Publishing.

Copyright © 2000 by SAMS Publishing.

All rights reserved. For sale in Mainland China only.

本书中文简体字版由美国 SAMS Publishing 公司授权人民邮电出版社出版。未经出版者书面许可，对本书任何部分不得以任何方式复制或抄袭。

版权所有，侵权必究。

内容提要

本书以流行的 Qt 库为对象，从程序设计角度全面介绍 Linux/UNIX 下基于 Qt 的图形界面程序开发方法。全书共分 5 部分：第一部分—— Qt 基础知识，介绍 Qt 基本知识、面向对象程序设计、槽和信号、Qt 构造块等内容。第二部分—— 重要的 Qt 部件，介绍常用 Qt 部件使用方法，以及怎样绘制图形和创建对话框等。第三部分—— 深入学习 Qt，介绍布局管理器、文件和目录、文本和常规表达式、容器类、图形、程序间通信等内容。第四部分—— Qt 编程技巧，介绍怎样编写 KDE 应用程序、使用 OpenGL 类绘图和创建 Netscape 插件等。第五部分—— 改善程序性能，介绍 Qt 程序的国际化、移植、调试等问题，以及怎样使用构造程序简单快捷地创建图形界面。附录部分给出了每章测验题答案和常用的 Qt 类描述。全书分为 24 个学时，每一学时内容均以前一学时为基础。

全书循序渐进，内容难度适中，实用性和操作性强，适用于 Linux/UNIX 和 Microsoft Windows 下的 GUI 程序开发人员学习使用。

关于作者

Daniel Solin 是 Solin Linux Consulting(www.solin.org)公司的总裁，这是一家专门从事 Linux 商业集成、Web 集成和图形接口的咨询公司。当空闲时，Daniel 还向当地的年轻人讲授 C/C++课程，使他们理解面向对象程序设计的优点。然而，自从他为美国 Macmillan 公司工作以来，大概半年时间，Daniel 已经放弃了其他项目。

Daniel 还参加了美国 Macmillan 公司的一些其他图书项目：*Slackware Linux Unleashed*、*Caldera OpenLinux Unleashed* 和 *Debian Linux Unleashed*。但是，他是一个非常执著的 Slackware 爱好者，因此他非常专心于 Slackware 项目。

Daniel 第一次接触 Linux 是在 1994 年。那时，他的家用 PC 上正运行一个古老的 UNIX 克隆版本 Minix，并因此爱上了这个芬兰产品 Linux。在使用 Linux 几年之后，Daniel 听说了 GUI 库：Qt。他便下载、编译、安装了该产品，并再次爱上这个产品。他的 Qt 项目之一是 Tarman，用于管理 Slackware tgz 包的图形接口。这是一个简单但非常有用的程序，因此，要想了解一个过去（20 世纪）的 GUI 程序，请访问 www.solin.org/tarman。

读者可以通过电子邮件 daniel@solin.org 与 Daniel 联系。

献　词

这本书献给 Linda，当我只盯着屏幕而几乎不做其他任何事情时，她是如此地理解我。我也非常感谢她看上去对我所谈论的工作非常感兴趣，尽管我知道她对这些根本就没有兴趣。

Daniel Solin

致　谢

我要感谢美国 Macmillan 公司所有辛苦工作和提供帮助的人们，没有他们的帮助和支持，这本书将不可能面世。

特别感谢本书的策划编辑 William Brown 在整个写作过程中所给予我的帮助。William 使我开始编写本书，并在写作过程中提供了很多帮助。因为这是我写的第一本书，我需要一些特别的帮助，William 总是为我提供这些帮助。

也要特别感谢本书的技术编辑，Clint McCarty 和 Tony Amico。他们的技术知识和经验很有价值。他们总是提供帮助并纠正我所做错的事。

感谢执行编辑 Rosemarie Graham 对本书出版过程的监督。她努力使工作尽可能地顺利推进，她的帮助对本书来说是无价的。

也要感谢美国 Macmillan 公司中我未实际接触的其他人员，包括出版者 Michael Stephens 和文字编辑 Bart Reed，他使我“不总是很好”的英语变得可以理解。我还要感谢 Michael 邀请我与他同写另一本书。

然而，美国 Macmillan 公司中还有很多我不知道名字的人，只要他们对本书有一点帮助，我都感谢他们。

我要感谢美国 Macmillan 公司的 Don Roche。他虽没有真正参与本书，但实际上是因为他我才获得编写本书的机会。Don 在 1999 年 5 月与我联系，希望我编写 *Slackware Unleashed* 一书中的部分内容。在那之后，他为我提供了一个令人激动的工作。如果不是 Don，我很可能从不会与美国 Macmillan 接触。感谢 Don!

还应该提到的是，在本书编写初期我得了流感，并因此使本书耽搁了一些时间。尽管这样，美国 Macmillan 公司的所有人员还一直鼓励我。

还要感谢 Troll Tech 的 Eirik Eng 让我使用他们的程序实例。

作　者

24 学时快速预览

学　时	标　题	内　容
1	Qt 简介	学习 Qt 库和 Qt 参考文档
2	面向对象程序设计	了解面向对象程序设计基本知识，学习为什么和怎样使用它
3	Qt 基础	创建第一个窗口，并向它添加按钮
4	槽和信号	研究 Qt 信号/槽机制，以连接部件事件
5	深入学习 Qt 构造块	创建滚动条和菜单，学习文件 I/O 操作和 QMainWindow 类
6	认识 Qt 部件的第 1 课	学习创建按钮、标签和表
7	认识 Qt 部件的第 2 课	学习使用选择部件、布局管理器、滑动框和微调框
8	认识 Qt 部件的第 3 课	学习使用文本输入域、列表视图和进程调
9	创建简单图形	了解 QPainter 类，处理颜色和打印图形
10	理解 Qt 对话框	研究 Qt 预定义对话框，学习创建自己的对话框
11	使用布局管理器	学习怎样使用布局管理器来放置部件
12	处理文件和目录	学习使用 Qt 类来处理文件和目录
13	文本和常规表达式	了解常规表达式，使用 QValidator 类
14	学习使用容器类	学习容器类怎样存储其他类对象
15	深入理解图形	了解更高级图形功能
16	程序通信	学习怎样使用剪贴板和怎样实现拖放功能
17	编写 KDE 应用程序的第 1 课	学习 KDE 开发基本知识，学习使用 KDE 的 HTML 功能
18	编写 KDE 应用程序的第 2 课	了解 KDE 核心库、用户接口库和文件操作库
19	使用 Qt 的 OpenGL 类	学习怎样在 Qt 程序中实现 OpenGL 功能
20	创建 Netscape 插件	学习怎样使用 Qt 创建 Netscape 浏览器插件
21	Qt 程序国际化	学习怎样使用翻译功能，怎样处理日期和时间值
22	可移植性	学习哪些 Qt 函数是可移植的和不可移植的
23	调试技术	了解 Qt 调试函数和宏
24	使用 Qt 构造程序	学习怎样使用 3 种 Qt 程序创建图形用户界面

前　言

这些年人们使用计算机的方式已经发生很大变化。最初是在一个黑屏幕上使用一些神秘的命令，但经过近 10 年的发展，已经或多或少进入图形环境，键盘的作用越来越小。今天，很少再有人考虑使用基于文本的程序。因此，对今天的用户来说，图形用户界面已经是成功开发应用程序所必需的。

这本书一步步地教你怎样创建易用的图形程序，从最基本的功能到最复杂的功能。本书中的 24 课包括逻辑解释、实例和指导，它们使得学习 Qt 编程变得更加简单。本书将教会你：

- 构造 Linux/UNIX 和 Microsoft Windows 下的图形界面应用程序；
- 安装 Qt 库和编写、编译、运行 Qt 程序；
- 用 KDE 编程；
- 理解面向对象程序设计（OOP）基本知识；
- 创建与用户交互所使用的对话框；
- 使用布局管理器；
- 创建和使用不同格式的图形文件；
- 实现剪贴板和拖放功能；
- 使用所有的 Qt 构造块——包括按钮、菜单、滑动框、滚动条……

有几个不同的工具包可用于创建 Linux 下的图形程序。但是，本书中你将学习的 Qt 是比较流行的一个。无论对于初学者还是专家，Qt 都是一个很好的选择，因为它具有面向对象的层次、好的结构、开发得很好的部件以及用于建图形界面以外的其他很多功能。Qt 也是众所周知的桌面环境 KDE 所使用的工具包，因此，它与 Gtk+工具包一起构成 Linux 下的图形程序标准。然而，与 Gtk+相比，Qt 的优点是它不仅适用于 UNIX/Linux，而且还可用于 Microsoft Windows。这意味着用在 Linux 下编写的 Qt 程序也是有效的 Microsoft Windows 程序。

阅读本书，你将对同时开发 Microsoft Windows 和 Linux/UNIX 环境下的图形程序是如此简单感到惊讶！

图标说明：



技巧：指出捷径和解决方法。



注意：用直观的方式阐明概念和过程。



警告：帮助你避免常规错误。

快速任务索引

操　作	节　号	页　码
安装 Qt 库	1.2	4
编译 Qt 程序	1.4	9
创建 Quit 按钮	3.2.3	37
创建用户槽	4.3	45
添加滚动条	5.1	53
创建按钮	6.1	69
在部件上书写文本	6.2	74
创建滑动框和微调框	7.3	93
创建文本输入域	8.1	99
创建进度条	8.3	106
实现打印功能	9.3	120
添加文件对话框	10.1.2	125
读取文件	12.1	155
实现文本验证功能	13.3	171
怎样装载和保存图像	15.2	189
实现剪贴板功能	16.1	199
实现拖放功能	16.2	205
将 Qt 程序翻译为不同语言	21.2	261
使用 gdb 调试 Qt 程序	23.3	290

目　录

第一部分　Qt 基础知识

第 1 学时　Qt 简介 …… 3

1.1　选择 Qt 库 …… 3
1.1.1　可移植性 …… 3
1.1.2　易用性 …… 3
1.1.3　运行速度 …… 4
1.2　安装 Qt 库 …… 4
1.2.1　编译和安装 Qt 源分发程序 …… 4
1.2.2　安装 Qt RPM 包 …… 7
1.3　一个简单的程序实例 …… 8
1.4　编译和运行 Qt 程序 …… 9
1.4.1　在 UNIX 系统下编译 …… 9
1.4.2　在 MS Windows 下使用 Visual C++编译 …… 10
1.5　使用 Qt Reference Document …… 11
1.6　小结 …… 14
1.7　问题与答案 …… 14
1.8　作业 …… 14
1.8.1　测验 …… 14
1.8.2　练习 …… 15

第 2 学时　面向对象程序设计 …… 17

2.1　理解类 …… 17
2.2　类继承 …… 23
2.3　Qt 如何使用 OOP …… 25
2.3.1　Qt 中使用类继承 …… 25
2.3.2　创建对象和访问方法 …… 26
2.4　小结 …… 28
2.5　问题与答案 …… 28
2.6　作业 …… 28

2.6.1 测验 …… 29
2.6.2 练习 …… 29

第 3 学时 Qt 基础 …… 31

3.1 创建第一个主部件 …… 31
3.2 向主部件中添加对象 …… 34
3.2.1 添加按钮 …… 34
3.2.2 添加标签 …… 36
3.2.3 添加退出按钮 …… 37
3.3 小结 …… 39
3.4 问题与答案 …… 39
3.5 作业 …… 39
3.5.1 测验 …… 40
3.5.2 练习 …… 40

第 4 学时 槽和信号 …… 41

4.1 理解信号和槽 …… 41
4.1.1 槽 …… 41
4.1.2 信号 …… 42
4.2 使用预定义信号和槽 …… 42
4.2.1 例 1——QSlider 和 QLCDNumber …… 42
4.2.2 例 2——QPushButton 和 QLineEdit …… 44
4.3 创建和使用用户信号和槽 …… 45
4.3.1 认识元对象编译器 …… 46
4.3.2 定位元对象编译器 …… 46
4.4 创建用户槽 …… 46
4.4.1 声明用户槽 …… 46
4.4.2 定义用户槽 …… 47
4.4.3 编译使用用户槽程序 …… 48
4.4.4 创建用户信号 …… 48
4.5 信号和槽的有趣功能 …… 50
4.5.1 避免不必要的信息 …… 50
4.5.2 信号和信号之间的连接 …… 50
4.5.3 断开槽和信号之间的连接 …… 50
4.5.4 使用 connect()函数时省略对象名称 …… 51
4.6 小结 …… 51
4.7 问题与答案 …… 51
4.8 作业 …… 52
4.8.1 测验 …… 52
4.8.2 练习 …… 52

第 5 学时　深入学习 Qt 构造块 …… 53
5.1　使用滚动条 …… 53
5.1.1　了解滚动条 …… 53
5.1.2　一个实际的例子 …… 55
5.2　添加菜单 …… 58
5.3　使用 QMainWindow 部件 …… 61
5.3.1　添加菜单、按钮和中心部件 …… 61
5.3.2　添加状态条 …… 64
5.4　小结 …… 65
5.5　问题与答案 …… 65
5.6　作业 …… 66
5.6.1　测验 …… 66
5.6.2　练习 …… 66

第二部分　重要的 Qt 部件

第 6 学时　认识 Qt 部件的第 1 课 …… 69
6.1　使用按钮 …… 69
6.1.1　按钮 …… 69
6.1.2　单选按钮 …… 71
6.1.3　复选按钮 …… 72
6.2　创建标签 …… 74
6.2.1　QLabel …… 74
6.2.1　QLCDNumber …… 75
6.3　表 …… 77
6.3.1　创建简单的网格 …… 77
6.3.2　添加文本和点击选择功能 …… 79
6.3.3　增加表头 …… 81
6.4　小结 …… 83
6.5　问题与答案 …… 83
6.6　作业 …… 84
6.6.1　测验 …… 84
6.6.2　练习 …… 84
第 7 学时　认识 Qt 部件的第 2 课 …… 85
7.1　选择部件 …… 85
7.1.1　列表框 …… 85
7.1.2　组合框 …… 86

7.2 部件布局 …… 88
7.2.1 QGroupBox 类 …… 88
7.2.2 QButtonGroup 类 …… 89
7.2.3 QSplitter 类 …… 89
7.2.4 QWidgetStack 类 …… 91
7.3 滑动框和微调框 …… 93
7.3.1 QSlider 类 …… 93
7.3.2 QSpinBox 类 …… 94
7.4 小结 …… 95
7.5 问题与答案 …… 96
7.6 作业 …… 96
7.6.1 测验 …… 96
7.6.2 练习 …… 97

第 8 学时 认识 Qt 部件的第 3 课 …… 99

8.1 文本输入域 …… 99
8.1.1 QLineEdit …… 99
8.1.2 QMultiLineEdit …… 100
8.2 理解列表视图 …… 101
8.3 进程条 …… 106
8.4 小结 …… 108
8.5 问题与答案 …… 108
8.6 作业 …… 108
8.6.1 测验 …… 108
8.6.2 练习 …… 109

第 9 学时 创建简单图形 …… 111

9.1 QPainter 类 …… 111
9.1.1 QPainter …… 111
9.1.2 设置绘图样式 …… 112
9.1.3 QPainter 绘图函数 …… 115
9.2 使用颜色 …… 118
9.2.1 管理颜色 …… 118
9.2.2 指定颜色 …… 119
9.3 用 Qt 打印图形 …… 120
9.4 小结 …… 121
9.5 问题与答案 …… 121
9.6 作业 …… 122
9.6.1 测验 …… 122
9.6.2 练习 …… 122

第 10 学时　理解 Qt 对话框 …… 123

10.1　预定义对话框 …… 123
10.1.1　颜色对话框 …… 123
10.1.2　文件对话框 …… 125
10.1.3　字体对话框 …… 126
10.1.4　消息对话框 …… 128
10.1.5　进度对话框 …… 129
10.2　创建用户对话框 …… 131
10.2.1　用 QDialog 创建用户对话框 …… 131
10.2.2　选项卡对话框 …… 134
10.3　小结 …… 137
10.4　问题与答案 …… 137
10.5　作业 …… 137
10.5.1　测验 …… 137
10.5.2　练习 …… 138

第三部分　深入学习 Qt

第 11 学时　使用布局管理器 …… 141

11.1　理解布局管理器 …… 141
11.2　使用布局管理器 …… 142
11.2.1　按行和列安排部件 …… 143
11.2.2　QGridLayout …… 145
11.3　理解嵌套布局管理器 …… 147
11.4　小结 …… 152
11.5　问题与答案 …… 152
11.6　作业 …… 153
11.6.1　测验 …… 153
11.6.2　练习 …… 153

第 12 学时　处理文件和目录 …… 155

12.1　使用 Qt 类读取文件 …… 155
12.2　使用 Qt 类读取目录 …… 158
12.3　使用 Qt 类读取文件信息 …… 160
12.4　小结 …… 164
12.5　问题与答案 …… 164
12.6　作业 …… 164
12.6.1　测验 …… 164

12.6.2 练习 …… 165
第 13 学时 处理文本和理解常规表达式 …… 167
13.1 常规表达式 …… 167
13.1.1 元字符 …… 167
13.1.2 转义序列 …… 168
13.2 预定义验证类 …… 169
13.2.1 QDoubleValidator 类 …… 169
13.2.2 QIntValidator 类 …… 171
13.3 创建用户验证类 …… 171
13.4 小结 …… 174
13.5 问题与答案 …… 174
13.6 作业 …… 175
13.6.1 测验 …… 175
13.6.2 练习 …… 175
第 14 学时 学习使用容器类 …… 177
14.1 Qt 容器类 …… 177
14.2 栈和队列 …… 178
14.2.1 用 QStack 类创建栈 …… 178
14.2.2 用 QQueue 类创建队列 …… 180
14.3 散列表 …… 181
14.4 数据缓存 …… 182
14.5 迭代 …… 184
14.6 小结 …… 185
14.7 问题与答案 …… 186
14.8 作业 …… 186
14.8.1 测验 …… 186
14.8.2 练习 …… 186
第 15 学时 深入理解图形 …… 187
15.1 动画 …… 187
15.2 装载和保存图像 …… 189
15.2.1 Qt 图像格式 …… 190
15.2.2 所支持的图像格式 …… 191
15.3 QPainter 转换函数 …… 193
15.3.1 图像缩放 …… 193
15.3.2 图像剪切 …… 194
15.3.3 图像旋转 …… 194

15.3.4 图像平移 …… 194
15.3.5 改变视窗 …… 195
15.3.6 设置窗口大小 …… 195
15.4 小结 …… 196
15.5 问题与答案 …… 196
15.6 作业 …… 196
15.6.1 测验 …… 197
15.6.2 练习 …… 197

第 16 学时 程序间通信 …… 199

16.1 剪贴板 …… 199
16.1.1 将剪贴板用于文本 …… 199
16.1.2 将剪贴板用于位图 …… 201
16.2 实现拖放功能 …… 205
16.3 小结 …… 208
16.4 问题与答案 …… 208
16.5 作业 …… 208
16.5.1 测验 …… 208
16.5.2 练习 …… 209

第四部分 Qt 编程技巧

第 17 学时 编写 KDE 应用程序的第 1 课 …… 213

17.1 KDE 程序设计基础 …… 214
17.1.1 安装 KDE …… 214
17.1.2 编写第一个 KDE 程序 …… 214
17.1.3 添加按钮、菜单、工具栏和状态栏 …… 215
17.2 使用 KDE 的 HTML 功能特点 …… 219
17.3 小结 …… 222
17.4 问题与答案 …… 222
17.5 作业 …… 223
17.5.1 测验 …… 223
17.5.2 练习 …… 223

第 18 学时 编写 KDE 应用程序的第 2 课 …… 225

18.1 KDE 核心库 …… 225
18.1.1 用 KAccel 类创建键盘快捷方式 …… 225
18.1.2 用 KPixmap 类管理图像 …… 226

18.1.3 用 KProcess 类启动子进程 …… 226
18.1.4 通过 KWM 类与 Window Manager 交互 …… 227
18.2 KDE 用户接口库 …… 228
18.3 KDE 文件操作库 …… 229
18.3.1 用 KDirDialog 类选择目录 …… 229
18.3.2 用 KFileDialog 类选择文件 …… 230
18.3.3 用 KFileInfo 类读取文件信息 …… 230
18.3.4 用 KFilePreviewDialog 预览文件 …… 231
18.4 其余 KDE 库 …… 232
18.5 小结 …… 232
18.6 问题与答案 …… 232
18.7 作业 …… 233
18.7.1 测验 …… 233
18.7.2 练习 …… 233

第 19 学时 使用 Qt 的 OpenGL 类 …… 235

19.1 建立 OpenGL 开发环境 …… 235
19.1.1 获取和安装 MESA …… 235
19.1.2 编译 Qt 的 OpenGL 扩展 …… 236
19.2 Qt 的 OpenGL 类 …… 236
19.2.1 QGLWidget——OpenGL 部件 …… 236
19.2.2 QGLContext——绘制 OpenGL 图形 …… 237
19.2.3 QGLFormat——设置环境显示格式 …… 238
19.3 编写、编译和运行基于 Qt 的 OpenGL 程序 …… 239
19.3.1 阅读代码 …… 239
19.3.2 编译和运行例子 …… 245
19.4 小结 …… 247
19.5 问题与答案 …… 247
19.6 作业 …… 247
19.6.1 测验 …… 247
19.6.2 练习 …… 248

第 20 学时 创建 Netscape 插件 …… 249

20.1 建立插件开发环境 …… 249
20.1.1 获得 Netscape Plugin SDK …… 249
20.1.2 编译 Qt 的 Netscape 插件扩展 …… 250
20.2 Qt 的 Netscape 插件类 …… 250
20.2.1 QNPlugin: 插件核心 …… 250
20.2.2 QNPInstance: 浏览器和插件之间的链接 …… 252
20.2.3 QNPWidget: 创建插件可视区域 …… 252

20.2.4 QNPStream: 从浏览器接收数据流 …… 253
20.3 创建第一个 Netscape 插件 …… 253
20.3.1 研究代码 …… 253
20.3.2 编译和安装插件 …… 255
20.3.3 测试插件 …… 256
20.4 小结 …… 257
20.5 问题与答案 …… 257
20.6 作业 …… 257
20.6.1 测验 …… 257
20.6.2 练习 …… 258

第五部分 改善程序性能

第 21 学时 Qt 程序国际化 …… 261

21.1 QString 的重要性 …… 261
21.2 创建翻译文件 …… 261
21.2.1 使用 tr()函数 …… 261
21.2.2 使用 findtr 实用程序提取翻译文本 …… 262
21.2.3 使用 msg2qm 实用程序创建二进制翻译文件 …… 264
21.2.4 用 mergetr 合并修改 …… 265
21.3 在程序中实现翻译功能 …… 265
21.4 处理日期和时间值 …… 267
21.4.1 使用 QDate 类处理日期值 …… 267
21.4.2 使用 QTime 类处理时间值 …… 269
21.4.3 使用 QDateTime 类处理日期时间值组合 …… 270
21.5 小结 …… 272
21.6 问题与答案 …… 272
21.7 作业 …… 272
21.7.1 测验 …… 272
21.7.2 练习 …… 273

第 22 学时 可移植性 …… 275

22.1 编写可移植的 Qt 应用程序 …… 275
22.1.1 使用 Qt 类 …… 275
22.1.2 遵守 POSIX 标准 …… 277
22.1.3 隔离平台相关的调用 …… 277
22.2 不可移植的 Qt 函数 …… 278
22.3 用 tmake 实用程序构造可移植项目 …… 279

22.3.1 获取和安装 tmake …… 279
22.3.2 用 tmake 创建编译文件 …… 280
22.4 用 progen 产生项目文件 …… 284
22.5 小结 …… 285
22.6 问题与答案 …… 285
22.7 作业 …… 286
22.7.1 测验 …… 286
22.7.2 练习 …… 286

第 23 学时 调试技术 …… 287

23.1 使用 Qt 调试功能 …… 287
23.1.1 qDebug()函数 …… 288
23.1.2 qWarning()函数 …… 288
23.1.2 qFatal()函数 …… 289
23.2 理解 Qt 调试宏 …… 289
23.2.1 ASSERT()宏 …… 289
23.2.2 CHECK_PTR()宏 …… 290
23.3 用 gdb 调试器调试 Qt 程序 …… 290
23.3.1 获取和安装 gdb …… 290
23.3.2 使用 gdb …… 291
23.4 命令行选项 …… 293
22.5 小结 …… 293
22.6 问题与答案 …… 294
22.7 作业 …… 294
22.7.1 测验 …… 294
22.7.2 练习 …… 294

第 24 学时 使用 Qt 构造程序 …… 295

24.1 QtEz …… 295
24.1.1 获取和安装 QtEz …… 295
24.1.2 用 QtEz 创建简单的 GUI …… 296
24.2 QtArchitect …… 302
24.2.1 获取和安装 QtArchitect …… 302
24.2.2 用 QtArchitect 创建简单的 GUI …… 303
24.3 EBuilder …… 305
24.3.1 获取和安装 EBuilder …… 305
24.3.2 用 EBuilder 创建简单的 GUI …… 306
24.4 小结 …… 308
24.5 问题与答案 …… 308
24.6 作业 …… 309

24.6.1 测验 …… 309
24.6.2 练习 …… 309

附录 A 测验题答案 …… 311

附录 B 常用 Qt 类 …… 325

第一部分

Qt 基础知识

第 1 学时　Qt 简介

第 2 学时　面向对象程序设计

第 3 学时　Qt 基础

第 4 学时　槽和信号

第 5 学时　深入学习 Qt 构造块

第 1 学时　Qt 简介

开始的这一学时简要介绍 Qt 库，我们将讨论 Qt 的优点——这也是选择 Qt 而不是其他 GUI 设计库（如 Motif、Gtk+、wxWindows 和 Xforms）的原因。

也将教你怎样使用 Qt Reference Documentation（Qt 参考文档）。在使用 Qt 时这是一个很重要的信息资源。尽管本书介绍了很多你需要了解的 Qt 知识，但它没有覆盖该库的所有内容。因此，你将经常需要查阅 Qt Reference Documentation。

最后，你将学习怎样创建、编译和运行一个非常简单的 Qt 程序，它向你介绍 Qt 程序实际开发过程，在后面学时中将集中学习这些内容。

1.1　选择 Qt 库

GUI 工具包（或 GUI 库）是构造图形用户界面（程序）所使用的一套按钮、滚动条、菜单和其他对象的集合。在 UNIX 系统里，有很多可供使用的 GUI 库，其中之一就是 Qt 库——一个基于 C++编程语言的工具包。由于 Qt 是基于 C++（而不是 C），速度块，易于使用，并具有很好的可移植性。所以，当需要开发 UNIX 和（/或）MS Windows 环境下的 GUI 程序时，Qt 是最佳选择。

1.1.1　可移植性

Qt 不只是适用于 UNIX，它同样适用于 MS Windows。如果你是一个以编程为生的专业程序员（因为你在阅读本书，所以我认为你不是，这里让我们假设你是），你的目标一定是吸引尽可能多的用户，以使他们有机会购买/使用你的产品。如果你的主要平台是 MS Windows，你很可能使用标准库——Microsoft Foundation Classes（MFC，Microsoft 基类）。但是，这样做你可能失去世界上几百万 UNIX 用户。相反，如果你的主要平台是 UNIX，你可能使用其他工具包，如 Gtk+或 Xforms。但这样将导致你失去几百万（如果不是几十亿的话）MS Windows 用户。因此，最好的方法是采用一个既适用于 UNIX，又适用于 MS Windows 的 GUI 工具包，是这样吗？当然，答案就是 Qt。

1.1.2　易用性

如前所述，Qt 是一个 C++工具包，它由几百个 C++类构成，你在程序中可以使用这些类。因为 C++是面向对象的编程（Object-Oriented Programming，OOP）语言，而 Qt 是基于 C++构造，所以，Qt 也具有 OOP 的所有优点。本书第 2 学时“面向对象编程”将向你介绍

更多的 OOP 知识。

如果你仍不相信 Qt 是最好的选择，可以选择使用其他工具包，之后，你会得出，哪一个工具包更适合你的结论。

1.1.3 运行速度

Qt 非常容易使用，且也具有很快的速度。这两方面通常不可能同时达到。当我们谈论其他 GUI 工具包时，易用常意味着低速，而难用则常意味着快速（或者从另一个方面讲，低速意味着易用，而快速则意味着难以使用）。但当谈论 Qt 时，其易用性和快速则是密不可分的。这一优点要归功于 Qt 开发者的辛苦工作，他们花费了大量的时间来优化他们的产品。

导致 Qt 比其他许多 GUI 工具包运行速度快的另一个原因是它的实现方式。Qt 是一个 GUI 仿真工具包，这意味着它不使用任何本地工具包作调用。Qt 使用各自平台上的低级绘图函数仿真 MS Windows 和 Motif（商用 UNIX 的标准 GUI 库），当然，这能够提高程序速度。其他适用于多种平台的工具包，如 wxWindows，则是使用 API 层或 API 仿真，这些方法均以不同的方式使用本地工具包，从而降低程序的运行速度。

1.2 安装 Qt 库

Qt 安装过程非常简单。这一节介绍怎样在 UNIX/Linux 系统上安装 Qt。要在 MS Windows 上安装 Qt 时，你必须从 Troll Tech 公司购买软件使用许可证，或者使用该软件的评估版本。更多信息请访问www.troll.no站点。

1.2.1 编译和安装 Qt 源分发程序

也许最好的操作方法是从www.troll.no站点下载最新的 Qt Free Edition（Qt 免费版本，编写本书时其最新版本为 2.0.2），然后自己编译它。这样做能够确保 Qt 库被正确安装，并且保证所安装的软件版本中包含来自 Troll Tech 公司的最新功能和改进。

分发程序文件存放在ftp.troll.no站点的/qt/source 子目录中。编写本书时，最新版本位于 ftp.troll.no/qt/source/qt-2.0.2.tar.gz，文件长度大约为 4MB。当下载分发程序时，要确认它是最新版本（其版本号最大）。图 1-1 所显示的 Netscape 窗口列出了ftp.troll.no站点中的 ftp 文件。

现在开始下载。当下载窗口从屏幕上消失后，下载完成，你便可以开始安装。

将你最新下载的文件移动到/usr/local 目录中。现在，从/usr/local 目录中执行下面命令：

```
# tar xvfz qt-2.0.2.tar.gz
```



注意，当你阅读本书时，新的 Qt 版本很可能已经发行。这意味着文件名可能不再是 qt-2.0.2.tar.gz，它可能被修改为 qt-2.1.0.tar.gz 或 qt-2.1.1.tar.gz 等之类的名称。

该命令将把文件解压到一个子目录中，目录的名称也决定于你所安装的软件版本。在这里，子目录名称为/usr/local/qt-2.0.2。当所有文件被解开之后，你便可以删除或移动分发程序文件：

```
# cd /usr/local
# rm qt-2.0.2.tar.gz
```

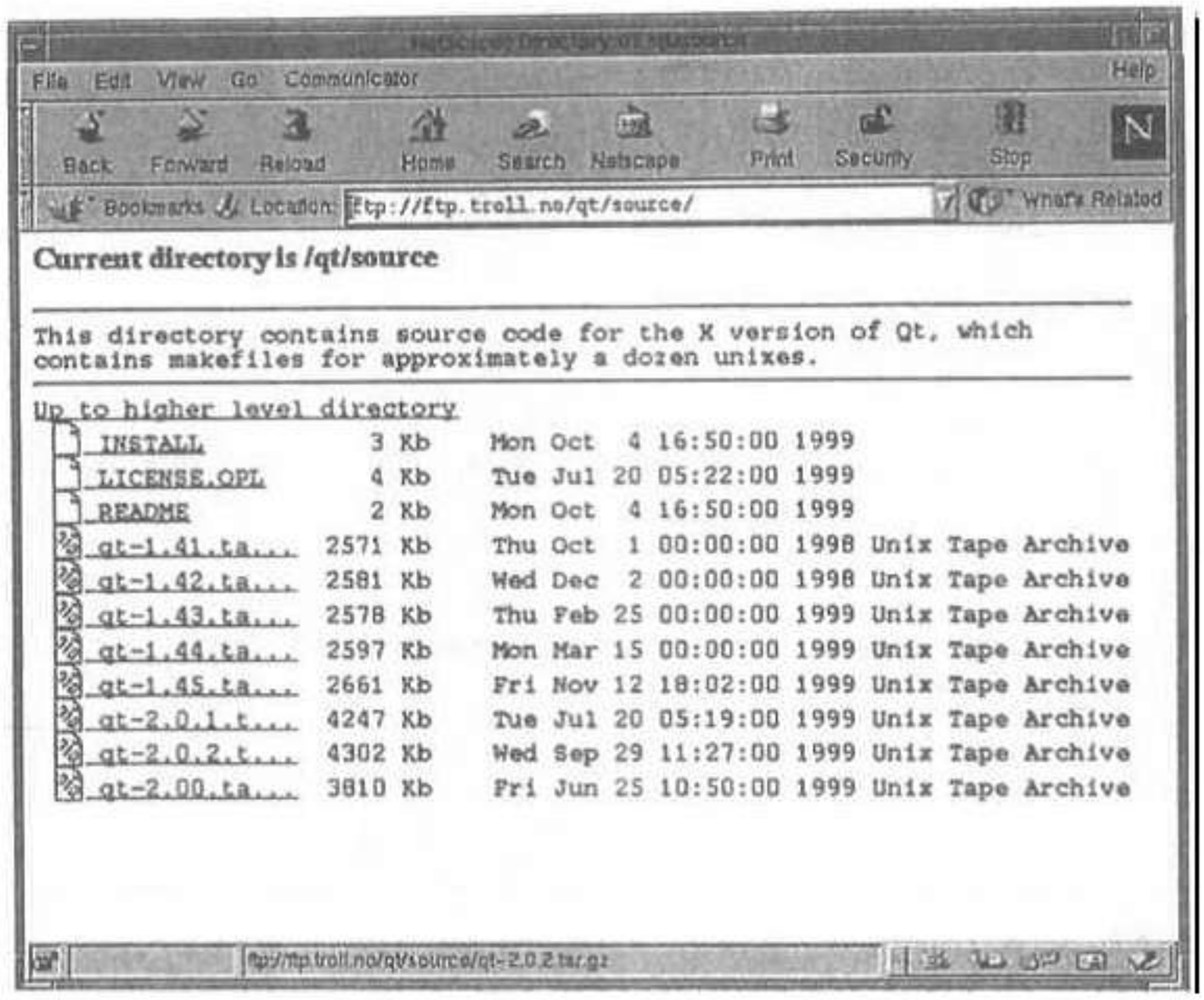


图 1-1　右击最新的分发程序文件，并选择 Save Link As 命令

下一步，你可以从以下两种操作中选择一种：将/usr/local/qt-2.0.2 目录移动到/usr/local/qt，或者从/usr/local/qt 创建一个符号连接，使它指向/usr/local/qt-2.0.2。无论选择哪一种操作，其效果完全相同（我本人喜欢符号连接，它使我能够很清楚地看到所安装的 Qt 软件版本号）。创建符号连接的方法为：

```
# ln -s /usr/local/qt-2.0.2 /usr/local/qt
```

需要移动（或重命名）目录时，可执行下面命令：

```
# mv /usr/local/qt-2.0.2 /usr/local/qt
```

下一步需要定义一些外壳变量。使用启动文件(究竟是哪个文件依赖于你所使用的外壳程序)定义外壳变量。如果使用 bash、ksh、zsh 或 sh，你可以在/etc/profile 文件内进行全局定义(这影响所有用户)，或者根据所使用的 Linux/UNIX 分发程序不同分别在$HOME/.profile ($HOME 表示你的主目录)或$HOME/.bash_profile 文件内定义个人变量(这只影响你自己)。当你决定将这些变量定义为全局变量(如果想要系统内所有用户都能够使用 Qt，这是一个好的选择)或个人变量后，将程序清单 1-1 中的代码添加到相应的文件即可。

如果你使用 csh 或 tcsh，则使用/etc/csh.login 进行全局设置，使用$HOME/.login 进行个人设置。将程序清单 1-2 中的代码添加到这些文件中。

程序清单 1-1　　为 bash、ksh、zsh 或 sh 定义 Qt 变量

```
1: QTDIR=/usr/local/qt
2: PATH=$QTDIR/bin:$PATH
3: if [ $MANPATH ]
4: then
```

```
 5:          MANPATH=$QTDIR/man:$MANPATH
 6: else
 7:          MANPATH=$QTDIR/man
 8: fi
 9: if [ $LD_LIBRARY_PATH ]
10: then
11:          LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
12: else
13:          LD_LIBRARY_PATH=$QTDIR/lib
14: fi
15: LIBRARY_PATH=$LD_LIBRARY_PATH
16: if [ $CPLUS_INCLUDE_PATH ]
17: then
18:          CPLUS_INCLUDE_PATH=$QTDIR/include:$CPLUS_INCLUDE_PATH
19: else
20:          CPLUS_INCLUDE_PATH=$QTDIR/include
21: fi
22: export QTDIR PATH MANPATH LD_LIBRARY_PATH LIBRARY_PATH
23: export CPLUS_INCLUDE_PATH
```

程序清单 1-2　　为 csh、tcsh 定义 Qt 变量

```
 1: if ( ! $?QTDIR ) then
 2:     setenv QTDIR /usr/local/qt
 3: endif
 4: if ( $?PATH ) then
 5:     setenv PATH $QTDIR/bin:$PATH
 6: else
 7:     setenv PATH $QTDIR/bin
 8: endif
 9: if ( $?MANPATH ) then
10:     setenv MANPATH $QTDIR/man:$MANPATH
11: else
12:     setenv MANPATH $QTDIR/man
13: endif
14: if ( $?LD_LIBRARY_PATH ) then
15:     setenv LD_LIBRARY_PATH $QTDIR/lib:$LD_LIBRARY_PATH
16: else
17:     setenv LD_LIBRARY_PATH $QTDIR/lib
18: endif
19: if ( ! $?LIBRARY_PATH ) then
20:     setenv LIBRARY_PATH $LD_LIBRARY_PATH
21: endif
22: if ( $?CPLUS_INCLUDE_PATH ) then
23:     setenv CPLUS_INCLUDE_PATH $QTDIR/include:$CPLUS_INCLUDE_PATH
24: else
25:     setenv CPLUS_INCLUDE_PATH $QTDIR/include
26: endif
```

当将程序清单 1-1 添加到/etc/profile 文件或$HOME/.profile 文件，或将程序清单 1-2 添加到/etc/csh.login 文件或$HOME/.login 文件后，应注销后重新登录，才能使这些设置生效。

做完这些之后，用 cd 命令转换到/usr/local/qt（它是一个符号连接或是一个真正的目录），你从这里开始实际编译。但是，首先必须运行 configure 脚本程序，这个脚本程序改变“编译文件”（这些文件告诉编译器怎样编译源程序），以便使它们适合你所使用的系统。

Configure 的输出类似于以下内容：

```
# ./configure

Build Type:  linux-g++-shared

Compile flags:  -I$(QTDIR)/src/3rdparty/zlib -I$(QTDIR)/src/3rdparty/libpng
Link flags:
GIF supports:  no

Creating makefiles…

Qt is now configured for building. Just run make.
To reconfigure, run make clean and configure.
```



注意，如果不带任何参数执行 configure 命令，编译文件将只被配置用于构造共享库，并不包含 GIF 图像支持。如果你也需要静态库，可向 configure 命令添加-static 选项。如果要支持 GIF，则需用-gif 选项执行 configure 命令。因此，为了包括 GIF 支持和构造静态库，需像下面这样执行 configure 命令：

```
# ./configure -gif -static
```

为了查看所有 configure 选项，输入下面命令：

```
# ./configure --help
```

当 configure 结束后，运行 make 开始编译：

```
# make
```

这时你会看到很多信息在屏幕上滚动，这些信息告诉你正在编译哪些内容，以及它们怎样被编译。你不必为这些信息烦心，这时你可以喝一杯咖啡，休息一下。当显示出下面信息时，说明编译已经完成：

```
The Qt library is now built in ./lib
The Qt examples are built in the directories in ./examples
The Qt tutorials are built in the directories in ./tutorial

Enjoy!  - the Troll Tech team
```

1.2.2　安装 Qt RPM 包

如果你使用基于 RPM（RedHat Package Manager）的 Linux 分发程序，如 RedHat、SusE、OpenLinux 或 TurboLlinux，Qt 很可能作为二进制 RPM 包包含在分发程序中。在提示符下或 X 终端仿真窗口内输入下面命令即可检查你的系统中是否包含 Qt：

```
# rpm -q qt
```

如果上面命令列出两个包——一个类似于 qt -<版本号>，另一个类似于 qt -devel -<版本

号>——说明你已经全部设置好。如果上面命令只列出 qt -<版本号>包，你则需要从分发程序光盘上安装 Qt 开发包。如果未列出任何一个，你则需要从分发程序光盘上安装这两个包。为了安装，使用 cd 命令切换到包含 rpm 分发程序的目录，并执行下面的命令：

```
# rpm -Uvh qt*
```

现在，你已经准备好，可以开始创建 Qt 应用程序，这是下一节将向你介绍的内容。

1.3 一个简单的程序实例

在程序员中，用一个 Hello World 程序向用户介绍一个新库已经变成一种传统做法。尽管 Hello World 程序非常简单（不管使用哪种库或工具包创建），但它们都能够给你一个很好的介绍，使你理解库的基本工作方式。在这一节，你将看到一个使用 Qt 创建的 Hello World 程序，该例子源代码如程序清单 1-3 所示。

程序清单 1-3 用 Qt 创建的 Hello World 程序

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qpushbutton.h>
4:
5: int main( int argc, char **argv )
6: {
7:     QApplication a( argc, argv );
8:
9:     QWidget mainwindow;
10:    mainwindow.setMinimumSize( 200, 100 );
11:    mainwindow.setMaximumSize( 200, 100 );
12:
13:    QPushButton helloworld( "Hello World!", &mainwindow );
14:    helloworld.setGeometry( 20, 20, 160, 60 );
15:
16:    a.setMainWidget( &mainwindow );
17:    mainwindow.show();
18:    return a.exec();
19: }
```

多数代码的作用不言自明。但为了彻底理解程序代码，我们还是逐行进行说明。

程序的第 1 行到第 3 行包含 3 个所谓的头文件，这些文件中包含各种声明，通过包含这些声明你可以在程序中使用库的某些内容。这里包含有 QApplication、QWidget 和 QPushButton 类的定义。因为头文件的使用不是 Qt 特有的，所以，你可能已经熟悉其用法。

第 5 行告诉你 main()函数定义从这里开始。所有 C/C++程序都必须包含一个 main()函数，main()函数总是首先被执行。该行为 main()函数声明两个参数——int argc 和 char **argv。这两个参数是 C++的内置功能，它们使命令行参数的使用更加简单。

第 7 行创建 QApplication 类对象。正如 C/C++程序只能包含一个 main()函数一样，Qt 程序中也只能包含一个 QApplication 对象。argc 和 argv 也用作 QApplication 对象构造函数参数，这使 Qt 处理命令行参数，如-geometry。

第 9 行创建一个新的 QWidget 对象，称做 mainwindow。你可以把 Qwidget 对像看作一

个窗口，在其上能够放置其他对象，如按钮。第 10 行和第 11 行将 mainwindow 的最小和最大尺寸均设置为 200 像素宽、100 像素高。当将窗口的最大尺寸和最小尺寸设置为相同值时，这意味着不能再调整这个窗口（mainwindow）的大小。

第 13 行创建 QPushButton 对象 helloworld，并直接调用 QPushButton 的一个构造函数。第一个参数告诉构造函数将按钮标签设置为“Hello World!”，第二个参数使 mainwindow 作为按钮的父窗口，这意味着按钮将被放置在 mainwindow 窗口上。

第 14 行设置 helloworld 几何尺寸，前两个参数告诉按钮的左上角在其父窗口（mainwindow）中的位置，它们相对于父窗口的左上角。因此，这里将把按钮左上角放置在相对于主窗口左上角右边 20 个像素和下边 20 个像素的位置。后两个参数设置按钮的宽度和高度（这里将其设置为 160 像素宽、60 像素高）。你可能认为这听起来有些难，但是放松一点，当你习惯它时，这将是最容易的。

第 16 行告诉 QApplication 对象将 mainwindow 设置为程序的主部件。当主部件被杀掉（关闭）时，整个程序的运行就结束。这一点对于多窗口程序非常有用。

第 17 行调用 mainwindow 的 show()函数。正如你可能猜到的一样，这意味着将把 mainwindow 显示在屏幕上。你不必调用 helloworld 的 show()函数，因为当其父窗口（mainwindow）的 show()函数被调用时，helloworld 被自动显示。

第 18 行将控制权从 main()函数传递给 Qt。在 exec()函数中，Qt 接收和处理用户及系统事件，并把这些事件传递给相应的窗口。当应用程序关闭时，exec()函数返回。图 1-2 为程序清单 1-3 的编译结果。



图 1-2　一个非常简单的 Qt 程序

这些并不难，是吗？事实上，从现在开始你所学习的所有内容都是基于这个例子。

1.4　编译和运行 Qt 程序

你已经安装了 Qt 库，并看到了一个简单的 Qt 程序，现在你将学习怎样编译和运行 Qt 程序。

1.4.1　在 UNIX 系统下编译

首先你要有需要编译的有效源代码，这里假设你使用程序清单 1-3 中的代码。将该代码保存到硬盘中的一个文件内，使用 cd 命令将当前目录切换为包含该文件的目录，并执行下面命令：

```
# g++ -lqt 01lst03.cpp -o 01lst03
```

这里，g++是 C++编译器。但是，在你的系统中，它也可能是 gcc、c++或其他类似的编

译器。-lqt 说明你希望与/usr/lib/qt/lib 目录中的 Qt 库 libqt.so 进行链接。01lst03.cpp 为你需要编译的源程序文件。-o 意味着其后面的字符串为你想调用的二进制文件（程序）。



当编译程序时，你可能会遇到下面错误信息：

```
can't load library 'libqt.so.2'
```

如果是这样的话，你要确保将程序清单 1-1 或 1-2 中的内容已经添加到启动文件中，并且在注销后重新登录以使这些修改生效。如果这些已经做过，但仍然得到错误信息，试着将/usr/local/qt/lib 添加到你的/etc/ld.so.conf 文件中，然后执行 ldconfig 程序：

```
# .ldconfig
```

如果仍然无效的话，执行下面命令检查 LD_LIBRARY_PATH：

```
# echo $LD_LIBRARY_PATH
```

如果该命令的输出中不包含/usr/local/qt/lib，则输入下面命令（如果你在使用 bash、ksh、zsh 或 sh）：

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/qt/lib
```

否则，如果你使用 csh 或 tcsh，则输入下面命令：

```
# setenv LD_LIBRARY_PATH /usr/local/qt/lib:$LD_LIBRARY_PATH
```

现在，再试着编译一次，如果有效，确保在你的任一个启动文件中将/usr/local/qt/lib 添加到 LD_LIBRARY_PATH。

接下来，运行程序，在包含你最新编译的程序目录中执行下面命令：

```
# ./01lst03
```

注意：你可以任意设置程序名称，在这个例子中，将程序命名为 01lst03 只是想说明它是从 01lst03.cpp 文件编译而来的。

1.4.2 在 MS Windows 下使用 Visual C++编译

为了能够在 MS Windows 系统下编译 Qt 程序，你需要做的第一件事是安装 Qt 库。这与安装所有其他 Windows 软件一样，因此这里不再进一步描述。但程序的实际编译操作可能有一点复杂，因为需要在 IDE（集成开发环境（Integrated Development Enviroment）——如 Microsoft 的 Visual C++）中修改一些设置。

当 Qt 库正确安装后，启动 Visual C++，点击 Project 菜单并选择 Settings。在打开的窗口中，选择 Link 选项，在 Object/Library Modules 文本域内输入以下内容：

```
qt.lib user32.lib gdi32.lib comdlg32.lib imm32.lib ole32.lib uuid.lib
 wsock32.lib
```

然后点击 OK 按钮关闭窗口，该窗口如图 1-3 所示。

你还需要告诉 Visual C++到哪里去查找 Qt 库和头文件。为了实现这一目标，从 Tools 菜单中选择 Options，在打开的窗口中选择 Directories。在目录配置菜单中查找 Show Directories For，并从菜单中选择 Include Files。在窗口的中部将显示一个大的区域，其中列出所有的包含目录。这时在最后一个目录的下面双击鼠标，并输入 c:\qt\include（如果你将 Qt 安装在 c:\qt 下）。之后，像添加包含文件目录一样，从同样的菜单中选择 Library Files，并添加目录

c:\qt\lib。最后，点击 OK 按钮使这些修改生效。该窗口如图 1-4 所示。

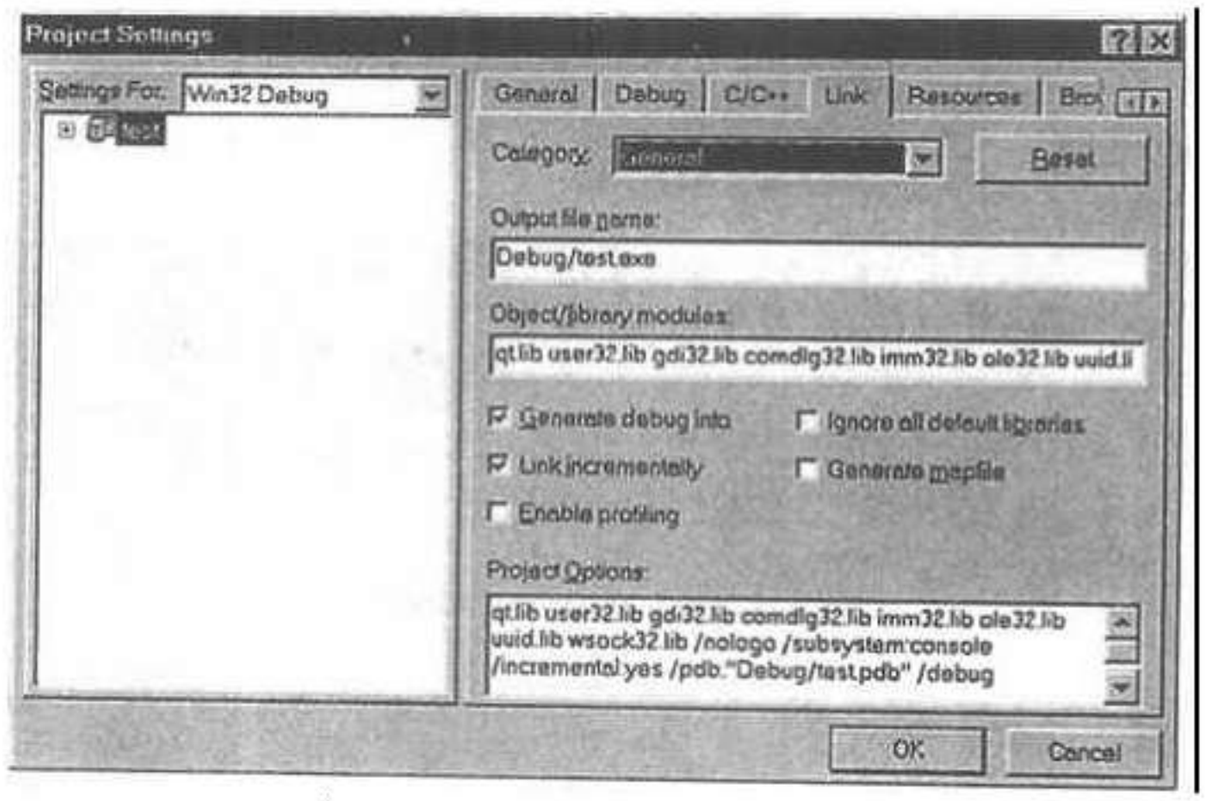


图 1-3　Visual C++的链接配置窗口

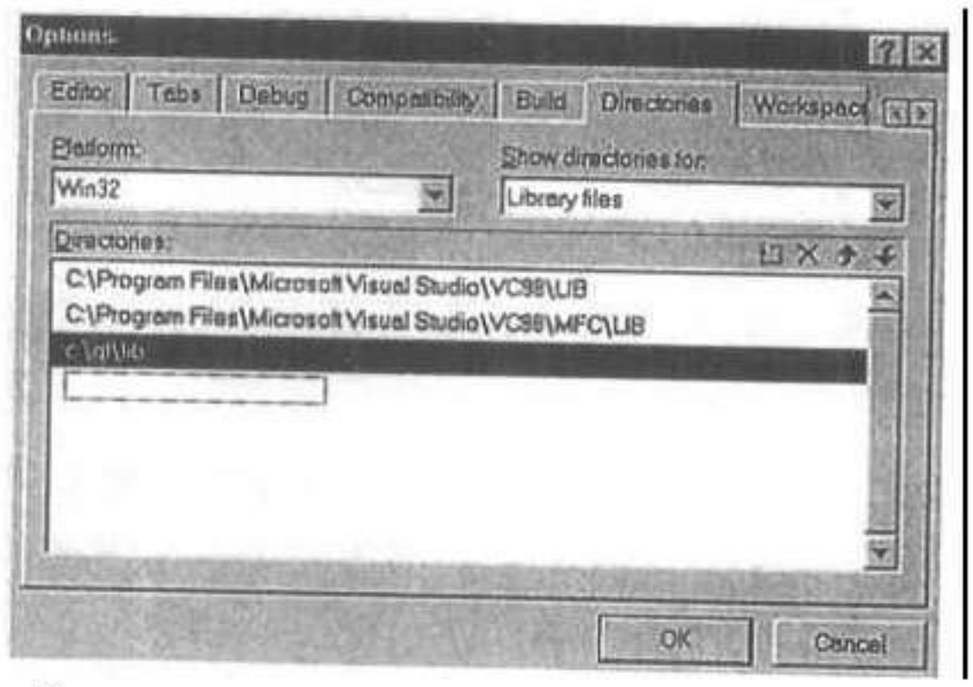


图 1-4　添加（和删除）Visual C++库和头文件的搜索目录

现在，你即可编译基于 Qt 的程序。用程序清单 1-3 例子进行编译测试，按 F7 键开始编译。如果遇到错误信息，检查前面步骤，保证各步骤都设置正确，也要确保 Visual C++和 Qt 变量都设置正确。

1.5　使用 Qt Reference Document

Qt Reference Document（Qt 参考文档）是使 Qt 变得如此流行的原因之一。它是一个非常出色的作品，当你需要检查库中的某些内容时，它是你的一个好助手。Qt Reference Document 结构非常好，因此，你能够非常容易地查找到你所需要的信息。你应该尽可能多地使用 Qt Reference Document，否则，编写 Qt 程序工作将变得更加困难和耗时。

既然已经有出色的文档可供使用，为什么还要写作这本关于 Qt 的书呢？尽管 Qt Reference Document 很好，但文档的大部分内容更像一个参考资料（正如其标题所指）而不是文档。如果你还不熟悉 GUI 开发，你可能需要更进一步的解释，而不只是 Qt Reference Document 所提供的内容，这正是本书所提供的。

Qt Reference Document 包含在 Qt 分发程序的 doc 子目录中，在这里是 HTML 版本，但从www.troll.no站点能够查找到可打印的 PostScript 版本。需要打开 Qt Reference Document 时，首先启动 Internet 浏览器，并将它指向 doc 子目录中的 index.html 文件，这时你将看到图 1-5 所示的 HTML 页面。

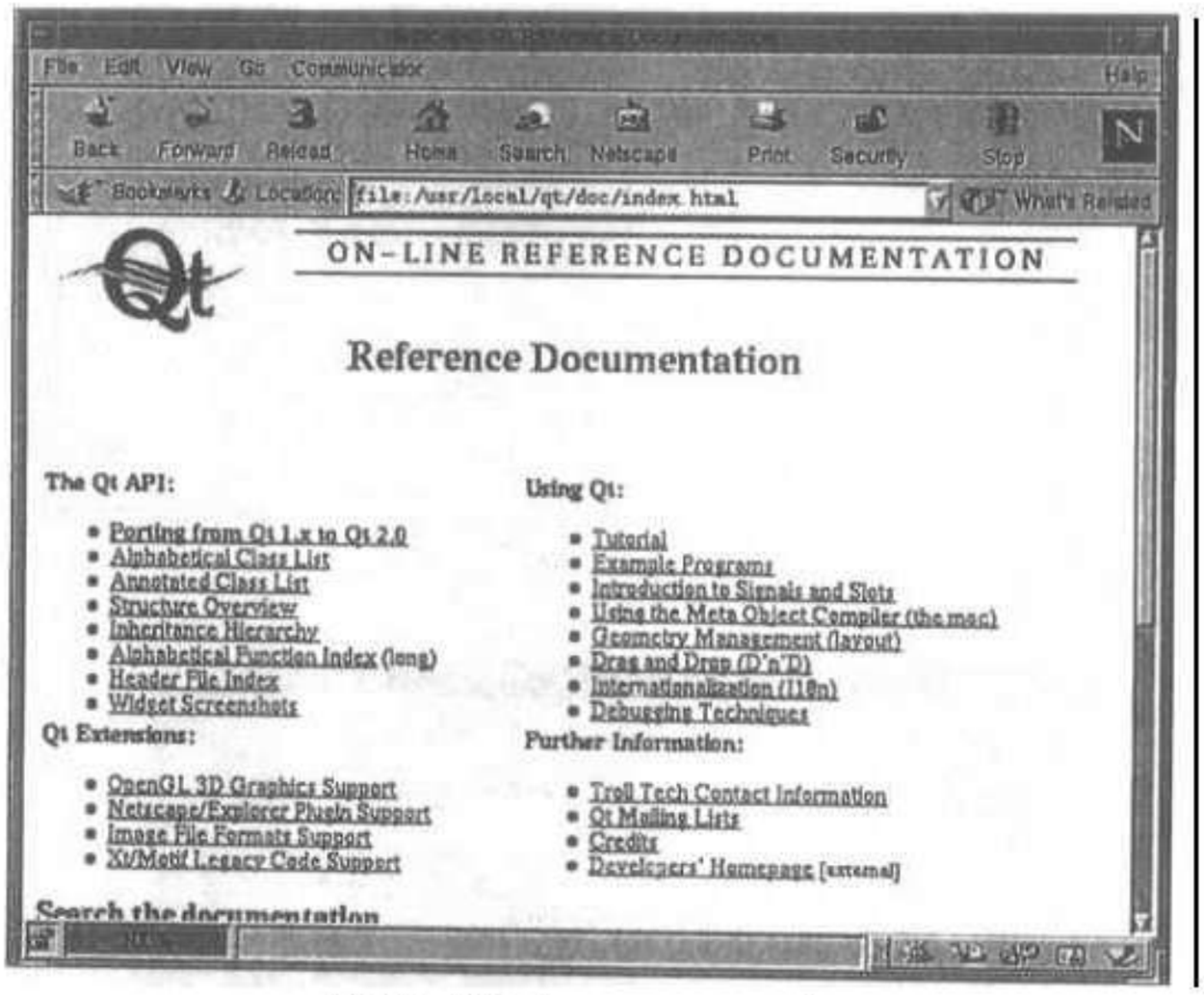


图 1-5 Qt Reference Document 主页

主页中的链接将你带到 Qt Reference Document 中的不同部分，所有这些部分构成对 Qt 库的完整参考。熟悉其中每一部分并能够很容易地查找到你所需要的东西非常重要。表 1-1 说明 Qt Reference Document 各部分的作用，希望你能够研究这张表，以便对 Qt Reference Document 组织方式有个总体了解。

表 1-1 Qt Reference Document 各部分内容介绍

部 分	描 述
Porting from Qt 1.x to Qt 2.0	这里你可以查找到怎样将基于 1.x 版本的 Qt 应用程序升级到 2.0 版本方面的信息
Alphabetical Class List	按字母顺序列出所有 Qt 类，每个类名提供到该类详细描述的链接
Annotated Class List	列出所有类，并对每个类有一个简短的描述。当你不熟悉 Qt，不知道某个任务应使用哪个类时，该信息就显得非常有用
Structure Overview	Qt 构造块综述，按类型排序。例如，当你想知道应选择哪个对话框时，就可以从这里查找
Inheritance Hierarchy	介绍类之间的相互关系
Alphabetical Function Index	Qt 所包含的所有函数列表
Header File Index	列出所有 Qt 头文件。这里，你还能察看包含的文件
Widget ScreenShots	所有 Qt 部件的拷屏图像，非常有用
Tutorial	Qt 指南——必读
Example Programs	Qt 所带的所有例子程序索引。当你需要一个实际例子时，这些程序非常有帮助
Introduction to Signals and Slots	信号和槽使你的程序能够与用户交互，没有它们，程序就没太大用处。这里，提供对它们的介绍信息
Using the Meta Object Compiler	元对象编译器使你能够更容易地创建信号和槽，这里介绍怎样使用它

续表

部　分	描　述
Geometry Management	阅读这部分（本书第 11 学时，“使用布局管理器”）内容后，在程序中能够更加容易地布置窗口
Drag and Drop	这部分介绍 Qt 程序怎样实现拖放操作
Internationalization	介绍怎样将程序国际化
Debugging Techniques	介绍怎样修订 Qt 程序错误
OpenGL 3D Graphics Support	OpenGL 是一种创建三维图形的方法，这里非常简要地讨论 Qt 对 OpenGL 编程方法的扩展
Netscape/Explorer Plugin Support	插件是 Internet 浏览器的辅助程序，浏览器使用插件处理某些文件，可以使用 Qt 编写插件程序，这就是这部分所介绍的内容
Image File Formats Support	想了解 Qt 应用程序怎样实现对某类图像格式的支持吗？如果是，请查看这部分内容
Xt/Motif Legacy Code Support	介绍 Qt 程序怎样使用旧的 Motif 代码

当你不熟悉 Qt 时，Annotated Class List 是最令你感兴趣的部分。如其名字所述，这部分包含所有 Qt 类的简短描述。当你需要查找一个类以执行某种特定任务时，它是一个很好的资源。当你知道所查找的类名，而想进一步了解该类信息时，Alphabetical Class List 将非常有用。

下面来看一个标准类参考。点击 Alphabetical Class List 链接，你会看到一个非常长的所有 Qt 类列表。点击其中的 QPushButton 链接，将向你显示该类的标准描述。Qt Reference Document 中的所有类描述与此类似（如图 1-6 所示）。

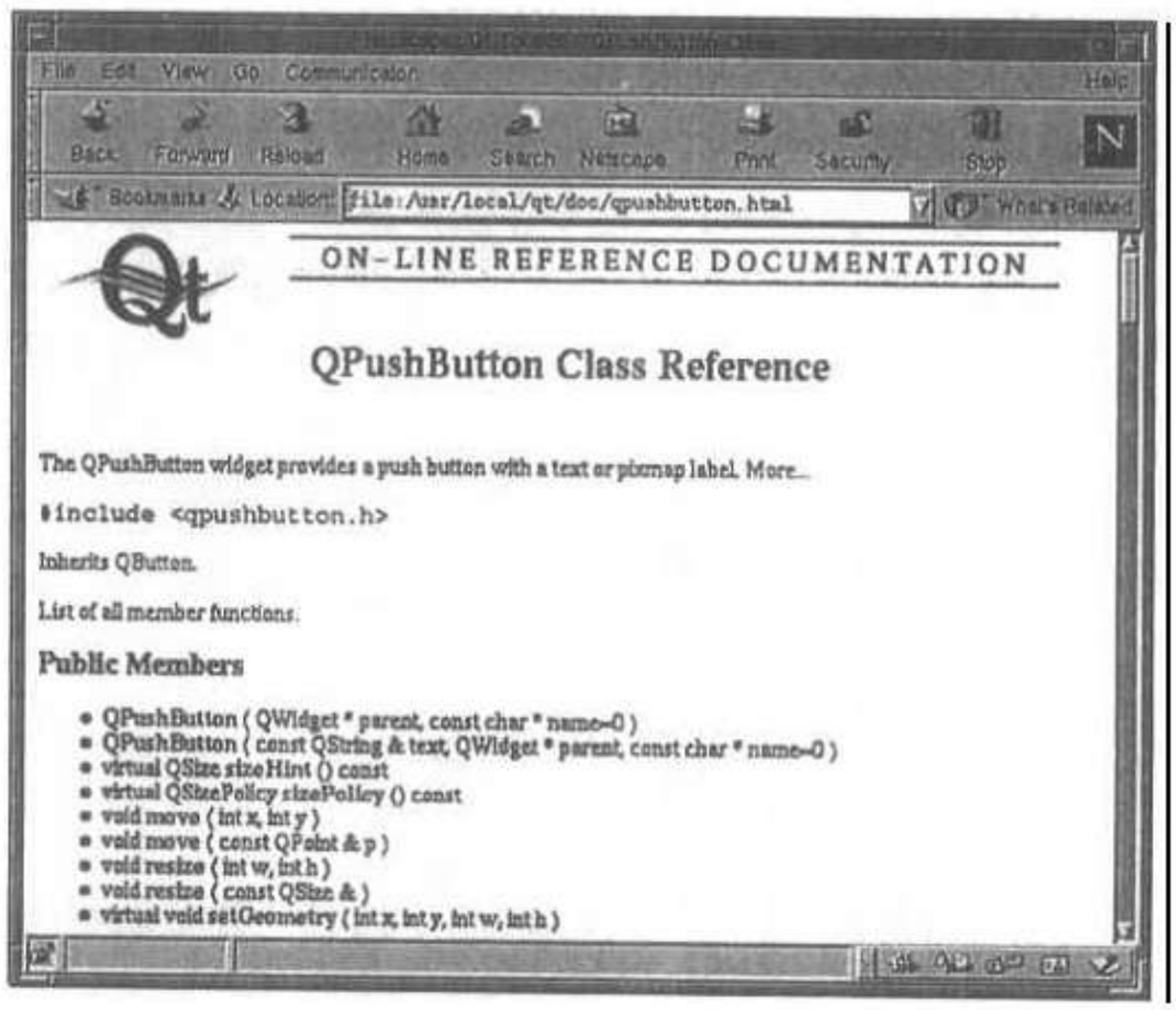


图 1-6　QPushButton 类描述页

在该页中你看到的第一件事就是该类的所有公有成员列表。每个函数实际上是到该函数更详细描述信息的一个链接。接下来是所有公共槽和保护成员（函数）列表。这些也是到槽和成员函数详细描述信息的链接。

在此之后是整个类的详细描述。在 QPushButton 类中，你将看到这个类是一个具有位图图像或文本标签的按钮部件。你也将看到该部件拷屏图像。

在当前这种水平下，文档的其他部分——Qt Extensions 和 Using Qt——可能用处不大。但当你了解库的工作方式和具有更多的 Qt 编程经验后，它们会变得更加有用。

1.6 小 结

到目前为止，你已经了解到 Qt 的优点，知道怎样安装它，和怎样编写一些简单的 Qt 程序。你也学习了怎样编译 Qt 程序。实际上，无论程序有多大或是多复杂，它们的编译过程是完全相同的。因此，从现在开始你将按照这一过程编译所有程序。但是，当项目较大时，将会有更多的文件需要编译。后面将向你介绍一个经常需要使用的工具 MOC。这一学时还向你介绍了 Qt Reference Document，你要确实理解怎样使用它和为什么要使用它。

完全理解这一学时中的所有内容是非常重要的，只有具备这些基本知识才能学习好后面学时中的内容。下一学时将介绍面向对象编程，在学完下一学时后，你将能够编写真正的面向对象的 Qt 程序。

1.7 问题与答案

问：为什么会出现未找到 make 命令的错误信息？

答：你需要安装 make，它包含在你的 Linux/UNIX 分发程序中。

问：为什么会出现未找到编译器的错误？

答：你必须安装一个 C++编译器，如 egcs，来编译 Qt 程序。

问：为什么在编译期间会出现一些库或包不存在的错误？

答：查找哪些库或包不存在，然后安装它们。如果你的分发程序中不包含这些库或包，可以通过 Internet 查找。如果你使用的是 Linux，www.freshmeat.net站点包含大量 Linux 程序资源。

问：我喜欢使用 Qt Reference Doument 打印版本，有这种版本吗？

答：是的，www.troll.no站点有其 PDF 版本。

1.8 作 业

完成下面的问题和练习将有助于你牢记本学时所学内容，并帮助你理解基本的 Qt 问题。

1.8.1 测验

1. 什么是 Qt？
2. 与其他同类产品相比，Qt 有哪些优点？
3. 什么是 QPushButton 类？
4. QWidget 类有什么用途？
5. a.setMainWidget(&mainwindow); 语句的意义是什么？

6. 从哪里查找 Qt Reference Document?

1.8.2　练习

1. 基于程序清单 1.3 编写一个程序，测试不同的几何参数。例如，使程序尺寸，包括按钮，变为原来的两倍。
2. 编译并运行你修改后的程序，如果其运行结果不是你所需要的样子，重新调整一些几何参数，并重新编译。
3. 为程序清单 1-3 中的程序增加另一个按钮，你可以通过扩大 QWidget 对象或缩小现有按钮大小来为新按钮腾出空间。然后重新编译和运行该程序。
4. 修改按钮上的文本。记注在每次修改源代码后都要重新编译。

第 2 学时　面向对象程序设计

如果你对程序设计有一点了解的话，你很可能听说过面向对象程序设计（Object-oriented Programming，OOP）。这是使程序设计变得更加简单的一种编程方法。OOP 代表很多新的程序设计思想，也是一种全新的程序设计方法。尽管它也基于一些“低级”程序设计功能（如 if、for、do 和 while 语句等），但 OOP 是一种新的程序设计方法。

OOP 使用类描述数据表。类与 C 语言中的结构非常类似，但类具有函数，并能够更明确地指出类成员（函数和变量）的访问方法。更详细内容将在这一学时的稍后部分介绍。

OOP 与语言无关，实际上，已经有很多面向对象的程序设计语言。但是，到目前为止，最流行的一种是 C++，它基于过去的 C 语言。Qt 是一个创建 GUI 程序的 C++类库。Qt 使用 OOP 的所有特性，为了使你能够成功使用 Qt，你需要一些基本的面向对象 C++程序设计知识。因此，你应该认真学习这一学时。



在我们开始之前，你应该明白这一学时中一些例子的部分代码属于我已经做过的一些事情——例如计算小汽车的最高速度和它的重量。对待这些不要太认真！

2.1　理解类

如前所述，C++使用类描述某种数据类型。现在我们学习怎样使用和创建简单类。

必须说明的是，C++类是一个非常大的话题，人们用整本书（远比这本书厚）的篇幅只介绍 OOP 和类。因此，这一节不能介绍关于类的所有内容，它只向你介绍需要了解的极少一部分内容。但是，这本书的目的是教你使用 Qt 创建 GUI 应用程序。所以，这部分内容也已足够。

一个类实例

假设你想创建一个描述小汽车的类——一个存储汽车重量、长度、宽度、最高速度、马力等数据的类。如果创建这样一个类，你得到一个存储这种信息的特殊数据类型。其定义为：

```
class car
```

```
{
public:
    int weight;
    int length;
    int width;
    int topspeed;
    int horsepower;
};
```

如果用过 C 结构，你会知道这个类的声明与结构声明非常相似。但是，眼下至少有两个不同的定义。第一，用 class 关键字代替 struct 关键字。第二，用 public:关键字指出其下面的变量能够在类里或类外访问。换句话说，你能够在程序中使用和修改这些变量。程序清单 2-1 是该类一个应用实例：

程序清单 2-1　　car 类应用实例

```
1: #include <iostream.h>
2:
3: class car
4: {
5: public:
6:     int weight;
7:     int length;
8:     int width;
9:     int topspeed;
10:    int horsepower;
11: };
12:
13: void main()
14: {
15:     //Create a object of the class:
16:     car a_car;
17:
18:     //Read the weight of the car from the keyboard:
19:     cout << "Define the car's weight!\n:";
20:     cin >> a_car.weight;
21:
22:     //Read the length of the car from the keyboard:
23:     cout << "Define the car's length!\n:";
24:     cin >> a_car.length;
25:
26:     //Read the width of the car from the keyboard:
27:     cout << "Define the car's width!\n:";
28:     cin >> a_car.width;
29:
30:     //Read the car's top speed from the keyboard:
31:     cout << "Define the car's top speed!\n:";
32:     cin >> a_car.topspeed;
33:
34:     //Read the car's horsepower from the keyboard:
35:     cout << "Define the car's horse power!\n:";
36:     cin >> a_car.horsepower;
37:
```

```
38:     //Print out that car's data:
39:     cout << "\n\nThis is the car's data:\n";
40:     cout << "\nWeight: " << a_car.weight;
41:     cout << "\nLength: " << a car.length;
42:     cout << "\nWidth: " << a_car.width;
43:     cout << "\nTop Speed: " << a_car.topspeed;
44:     cout << "\nHorse Power: " << a_car.horsepower;
45:     cout << "\n";
46: }
```

这个例子只是像使用结构一样使用类。但是，如果你想使类更加“智能”一点的话，则需要使用类的某些特有功能对它进行扩展。例如，如果想使该类能够做一些基于小汽车数据的计算，你就需要向类添加能够执行计算的成员函数。不能向结构添加函数，但可以向类添加。这个例子如程序清单 2-2 所示。

程序清单 2-2　　具有成员函数的 car 类

```
 1: #include <iostream.h>
 2:
 3: class car
 4: {
 5: public:
 6:      car();
 7:     void printdata();
 8:     void printweight();
 9:     void printtopspeed();
10: private;
11:     int length;
12:     int width;
13:     int horsepower;
14: };
15:
16: car::car()
17: {
18:     //Read the length of the car from the keyboard:
19:     cout << "Define the car's length!\n:";
20:     cin >> this->length;
21:
22:     //Read the width of the car from the keyboard:
23:     cout << "Define the car's width!\n:";
24:     cin >> this->width;
25:
26:     //Read the car's horsepower from the keyboard:
27:     cout << "Define the car's horse power!\n:";
28:     cin >> this->horsepower;
29: }
30:
31: void car::printdata()
32: {
33:     //Print out that car's data:
34:     cout << "\n\nThis is the car's data:\n";
35:     cout << "\nLength: " << this->length;
36:     cout << "\nWidth: " << this->width;
37:     cout << "\nHorse Power: " << this->horsepower;
38:     cout << "\n";
39: }
```

```
40:
41: void car::printweight()
42: {
43:     //Print out the car's weight to the screen:
44:     cout << "Weight: " << (this->length * this->width * 100 );
45: }
46:
47: void car::printtopspeed()
48: {
49:     //Print out the car's top speed to the screen:
50:     if( this->horsepower <= 200 )
51:     {
52:         cout << "\nTop Speed: " << (this->horsepower * 1.2 ) << "\n";
53:     }
54:     if( this->horsepower > 200 )
55:     {
56:         cout << "\nTop Speed: " << (this->horsepower * 0.8 ) << "\n";
57:     }
58: }
59:
60: void main()
61: {
62:     //Create an object of the class and call the member functions:
63:     car a_car;
64:     a_car.printdata();
65:     a_car.printweight();
66:     a_car.printtopspeed();
67: }
```

这个程序清单所介绍的一些新程序设计方法需要进行解释。这些方法在下面进行讨论。它们是面向对象 C++程序设计的精华，因此，你应该确实理解这些讨论。

● 首先你在类声明中看到一个非常奇怪的函数，其名称为 car()（永远与类名相同），它不带任何参数，也没有返回类型。这个函数称做构造函数。当创建类对象时调用该函数。在程序清单 2-2 中，构造函数在下面一行中被调用：

```
car a_car;
```

在 public:下声明的构造函数和其他成员函数使它们能够在类外部代码中被访问。如果不在 public:下声明，你将不能在 main()函数中调用它们。

● 数据类型（变量）声明在 private:关键字下。这使得只能在类里才能访问这些变量（也就是说，你不能在类外修改和访问数据）。这样做使程序更加安全，更少出错。这种编程方法称做数据隐藏，它被 C++程序员所广泛使用。

● 你可能不熟悉成员函数声明方法。使用::运算符说明函数属于哪个类（在这里为 car::）。

● 使用 this 指针访问当前类成员。this 指针代表对象，在编写类定义时，你还不知道该对象的名称。但是，在这个例子中，你可以省略 this 指针而只输入变量名；那么，程序将假定你所指变量属于当前对象（我包含它是想说明怎样使用它）。但是，当需要访问一个还没有创建的父对象时，this 指针将变得非常有用。是不是听起来有点复杂？如果是的话，那么找一本好的 C++方面的图书，那里会有更进一步的解释。



要确实理解怎样从类和结构中访问函数和变量。如果类对象或结构是一个真正的对象，使用点(.)运算符；如果对象是一个指针，则使用->运算符。



程序清单 2-2 所采用的方法使实际数据变为私有，因此，只能通过成员函数访问这些数据，这是 OOP 与过程化编程相比更加安全和容易使用的原因之一。因为只能在类里处理数据，因此，这些数据发生意外的可能性就大大降低。

前面已经提到，你可以为类增加构造函数，使它在每次创建类对象时处理所有你想做的工作。C++也包含一个析构函数。它用于释放构造函数所分配的内存，当对象完成或被删除时执行析构函数。析构函数必须命名为~<类名>()。因此，在当前例子中，析构函数应被命名为~car()。之后，你可以像定义其他函数一样定义析构函数，并使用 delete 关键字释放内存。然而，这需要使用动态内存分配方法。程序清单 2-3 对程序清单 2-2 进行了修改，使它包含一个析构函数。

程序清单 2-3　　具有析构函数的 car 类

```
 1: #include <iostream.h>
 2:
 3: class car
 4: {
 5: public:
 6:      car();
 7:     -car();
 8:     void printdata();
 9:     void printweight();
10:     void printtopspeed();
11: private:
12:     int *length;
13:     int *width;
14:     int *horsepower;
15: };
16:
17: car::car()
18: {
19:     //Allocate memory dynamically for the data.
20:     length = new int;
21:     width = new int;
22:     horsepower = new int;
23:
24:     //Read the length of the car from the keyboard:
25:     cout << "Define the car's length!\n:";
26:     cin >> *this->length;
27:
```

```
28:     //Read the width of the car from the keyboard:
29:     cout << "Define the car's width!\n:";
30:     cin >> *this->width;
31:
32:     //Read the car's horsepower from the keyboard:
33:     cout << "Define the car's horse power!\n:";
34:     cin >> *this->horsepower;
35: }
36:
37: car::~car()
38: {
39:     //This function will be called when to object is finished,
40:     //we free up memory with the delete keyword.
41:     delete length;
42:     delete width;
43:     delete horsepower;
44: }
45:
46: void car::printdata()
47: {
48:     //Print out that car's data:
49:     cout << "\n\nThis is the car's data:\n";
50:     cout << "\nLength: " << *this->length;
51:     cout << "\nWidth: " << *this->width;
52:     cout << "\nHorse Power: " << *this->horsepower;
53:     cout << "\n";
54: }
55:
56: void car::printweight()
57: {
58:     //Print out the car's weight to the screen:
59:     cout << "Weight: " << ( (*this->length) * (*this->width) * 100 );
60: }
61:
62: void car::printtopspeed()
63: {
64:     //Print out the car's top speed to the screen:
65:     if( *this->horsepower <= 200 )
66:     {
67:         cout << "\nTop Speed: " << ( (*this->horsepower) * 1.2 ) << "\n";
68:     }
69:     if( *this->horsepower > 200 )
70:     {
71:         cout << "\nTop Speed: " << ( (*this->horsepower) * 0.8 ) << "\n";
72:     }
73: }
74:
75: void main()
76: {
77:     //Create an object of the class and call the member functions:
78:     car a_car;
79:     a_car.printdata();
80:     a_car.printweight();
81:     a_car.printtopspeed();
82: }
```

正如你所看到的，new 和 delete 关键字用于分配和释放内存。注意，你需要使用*运算

符访问实际数据。如果省略*运算符，程序将只查找内存地址（因为你使用指针）。这是基本的 C 知识，因此，你应该已经掌握它。

2.2　类继承

关于 OOP 的另一个重要知识是它使你能够重用旧的代码。类继承使你在旧的已经测试过的类的基础上构造新类成为可能。你可以向类增加所需要的属性，但仍可使用旧的稳定的代码。

下面举个例子看类继承怎样工作。我们修改上一节中的类，使它也能够存储关于卡车的信息。这辆卡车碰巧是用于运输矿石的，你需要存储卡车的载重量信息。下面是该例子的程序清单：

程序清单 2-4　　基于 car 类的 truck 类

```
#include <iostream.h>

//First, we need to include the code for the car class:
class car
{
public:
     car();
    ~car();
    void printdata();
    void printweight();
    void printtopspeed();
private:
    int *length;
    int *width;
    int *horsepower;
};

car::car()
{
    //Allocate memory dynamically for the data.
    length = new int;
    width = new int;
    horsepower = new int;

    //Read the length of the car from the keyboard:
    cout << "Define the length!\n:";
    cin >> *this->length;

    //Read the width of the car from the keyboard:
    cout << "Define the width!\n:";
    cin >> *this->width;

    //Read the car's horsepower from the keyboard:
    cout << "Define the horse power!\n:";
    cin >> *this->horsepower;
}
```

```
38: car::-car()
39: {
40:     //This function will be called when to object is finished,
41:     //we free up memory with the delete keyword.
42:     delete length;
43:     delete width;
44:     delete horsepower;
45: }
46:
47: void car::printdata()
48: {
49:     //Print out that car's data:
50:     cout << "\n\nThis is the data:\n";
51:     cout << "\nLength: " << *this->length;
52:     cout << "\nWidth: " << *this->width;
53:     cout << "\nHorse Power: " << *this->horsepower;
54:     cout << "\n";
55: }
56:
57: void car::printweight()
58: {
59:     //Print out the car's weight to the screen:
60:     cout << "Weight: " << ( (*this->length) * (*this->width) * 100 );
61: }
62:
63: void car::printtopspeed()
64: {
65:     //Print out the car's top speed to the screen:
66:     if( *this->horsepower <= 200 )
67:     {
68:         cout << "\nTop Speed: " << ( (*this->horsepower) * 1.2 ) << "\n";
69:     }
70:     if( *this->horsepower > 200 )
71:     {
72:         cout << "\nTop Speed: " << ( (*this->horsepower) * 0.8 ) << "\n";
73:     }
74: }
75:
76: //Then we create a new class based upon car:
77: class truck : public car
78: {
79: public:
80:     truck();
81:     ~truck();
82:     void printoreload();
83: private:
84:     int *ore_load;
85: };
86:
87: truck::truck()
88: {
89:     //Allocate memory for our new data member:
90:     ore_load = new int;
91:
92:     //Read the ore load from the keyboard:
93:     cout << "Define the ore load!\n:";
94:     cin >> *this->ore_load;
95: }
```

```
 96:
 97: truck::-truck()
 98: {
 99:     //Free up memory allocated in the constructor:
100:     delete ore_load;
101: }
102:
103: void truck::printoreload()
104: {
105:     //Print out the truck's ore load:
106:     cout << "Ore load: " << *this->ore_load << "\n";
107: }
108:
109: void main()
110: {
111:     //Create an object of the class and call the member functions:
112:     truck a_truck;
113:     a_truck.printdata();
114:     a_truck.printweight();
115:     a_truck.printtopspeed();
116:
117:     //We also want to call the truck specific function printoreload():
118:     a_truck.printoreload();
119: }
```

当运行这个程序时，你很快会发现 car 类的构造函数也被执行。因为这样，所以你不必完全重写一个新的构造函数来从键盘读取所有的数据。相反，你只需另外创建一个 truck 类具有的与卡车特有属性相关的构造函数。因此，你可以节省大量的工作量。

你不必创建那些 car 类中已经创建的变量，但不是那些需要在新类中扩展使用的变量。也不需要重新定义用于计算车辆最高速度和重量的成员函数。

接下来，我们来讨论下面一行代码：

```
class truck : public car
```

这定义新类继承哪个类（这里为 car）。它使用 C++程序员熟悉的公有继承方法，这意味着基类中的公有成员在新类中仍为公有，基类中的私有成员在新类中仍为私有。如果将 public 关键字改为 private，则将使用私有继承。私有继承使基类中的所有函数（包括公有函数）在新类中都变为私有。但是，公有继承是目前使用最多的继承方法，Qt 中也是这样。

2.3　Qt 如何使用 OOP

Qt 几乎包含 OOP 的所有内容，这一节举几个例子说明 Qt 怎样使用 OOP。尽管 Qt 是一个完全面向对象的类库，但为了掌握它你不必了解 OOP 的所有内容。这一节介绍完全理解 Qt 库所需要的关键知识。

2.3.1　Qt 中使用类继承

本学时前面部分已经介绍了怎样在已有代码基础上使用类继承构造应用程序。创建 Qt 应用程序时，你会更多地用到类继承。事实上，编写 Qt 应用程序的大部分工作是在基于已

有的 Qt 类编写用户类。程序清单 2-5 就是这样一个非常简单的例子。

程序清单 2-5　　一个简单 Qt 类继承实例

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3:
4: class myclass : public QWidget
5: {
6: public:
7:     myclass();
8: };
9:
10: myclass::myclass()
11: {
12:     this->setMinimumSize( 200, 200 );
13:     this->setMaximumSize( 200, 200 );
14: }
15:
16: int main( int argc, char **argv )
17: {
18:   QApplication a( argc, argv );
19:   myclass w;
20:   a.setMainWidget( &w );
21:   w.show();
22:   return a.exec();
23: }
```

这个例子与前面的一个例子功能相同。这里使用公有继承继承 QWidget 类成员。所创建的新构造函数用于定义窗口（类）的最大和最小尺寸。这里也可以省略 this 指针，使用它只是使程序更加清晰。程序清单 2-5 的运行结果如图 2-1 所示。

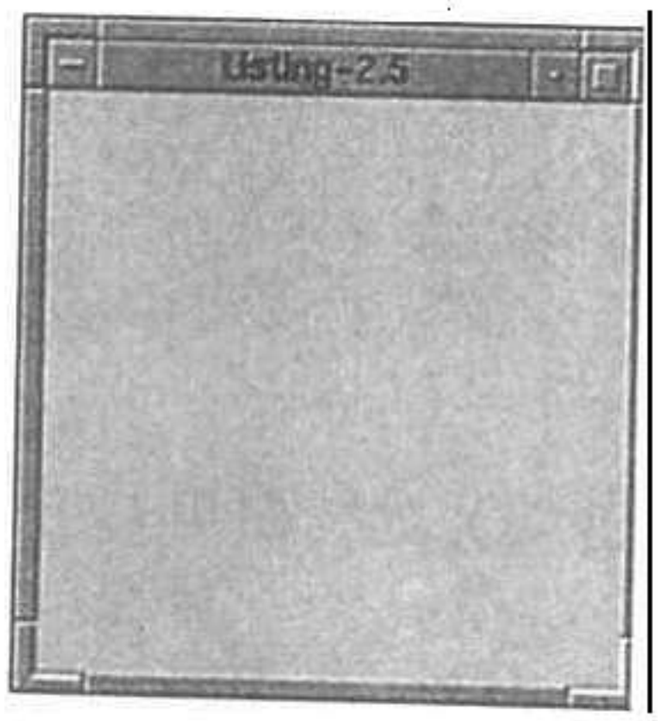


图 2-1　一个简单的基于 Qwidget 的类

2.3.2　创建对象和访问方法

因为 Qt 是一个类库，所以你必须知道怎样去创建它所提供的类对象。你已经在第 1 学时“Qt 简介”中看到了这样一个例子。但是，当创建你自己的类时，情况就有一点不同。事实上，在这一学时前面部分已经介绍了这种方法。但是，现在我们举一个使用 Qt 类的例

子。此外，你还应知道怎样访问包含在类中的方法（成员函数）。这个例子代码如程序清单 2-6 所示。它以注释方式帮你理解面向对象的 Qt 程序设计方法。

程序清单 2-6　　一个面向对象的 Qt 程序设计实例

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qpushbutton.h>
4:
5: //First, we describe the class; which method
6: //and other members it shall include.
7: class myclass : public QWidget
8: {
9: public:
10:     myclass();
11: private:
12: //Locate memory-space for a QPushButton object:
13:     QPushButton *b1;
14: };
15:
16: //Define the constructor:
17: myclass::myclass()
18: {
19:     //We leave out the this-pointer, the default
20:     //is the current object anyway:
21:     setMinimumSize( 200, 200 );
22:     setMaximumSize( 200, 200 );
23:
24:     //Allocate memory for the QPushButton object.
25:     //We pass two arguments to the class's
26:     //constructor. The first is the label of the
27:     //button, the second is the button's parent
28:     //widget. Here the this-pointer is really
29:     //useful!
30:     b1 = new QPushButton( "Hello", this );
31:     //Here we access the method setGeometry()
32:     //to set the geometry of the button.
33:     b1->setGeometry( 20, 20, 160, 160 );
34: }
35:
36: int main( int argc, char **argv )
37: {
38:     //Create an QApplication object:
39:       QApplication a( argc, argv );
40:       //Create an object of our new class:
41:       myclass w;
42:     //Access the QApplication::setMainWidget
43:     //method:
44:       a.setMainWidget( &w );
45:     //Call the show() method of w (myclass):
46:       w.show();
47:     //Pass control to the Qt library:
48:       return a.exec();
49: }
```

你现在对面向对象的 Qt 程序设计方法应该有一个基本的了解。图 2-2 为程序清单 2-6 的运行结果。



图 2-2 以面向对象方式编写的一个非常简单的 Qt 程序

2.4 小 结

这一学时介绍了许多新的复杂知识。如果你认为是这样的话，那么应该找一本好的 C++方面的图书来阅读，以帮助自己完全理解面向对象程序设计方法。尽管成功使用 Qt 库并不需要成为一个 OOP 专家，但是，花点时间学习 C++语言中的 OOP 知识将使你能够更容易地理解 Qt 库中的复杂内容。当你能够熟练使用 OOP 时，很快就会发现使用 C++所提供的新功能给你带来的好处。

2.5 问题与答案

问：我的编译器不能编译 OOP 代码，它显示缺少.h 文件，这是为什么？

答：你需要 OOP 编译器，如 egcs，来编译 OOP 程序，你还需要安装 C++库和头文件，你的编译器或分发程序带有这些文件。

问：当我编译一个面向对象程序时，为什么会出现构造函数返回类型错误消息？

答：你可能忘了在你所声明类的最后一个括号（}）后添加分号（;）。

问：当我编译 Qt 程序时，出现许多缺少参照错误，这是为什么？

答：你可能在编译程序时忘了添加-lqt 选项。

问：我遇到一条 base operand of '->' has non-pointer type 'car'错误消息，这是什么原因？

答：如果对象是一个指针，你应该使用->运算符访问其成员。否则，使用.运算符。

问：当编译程序时，我遇到一条 initializing non-const 'bool &' with 'int *' will use a temporary 警告信息，这是为什么？

答：你可能在访问一个指针变量，你需要使用*运算符去访问其实际数据。

2.6 作 业

完成下面的问题和练习将有助于你牢记本学时所学内容，并帮助你理解 OOP 的实现机

制。

2.6.1　测验

1. OOP 代表什么意思?
2. 什么是类?
3. 什么是对象?
4. 什么是方法?
5. 什么是类继承?
6. 使用 Qt 时为什么需要 OOP 知识?

2.6.2　练习

1. 编写一个类描述 Internet 网络冲浪者，该类记录冲浪者的性别、年龄、教育程度和每周上网时间，使用构造函数从键盘读取信息。
2. 修改上一个练习中的程序，使它使用动态内存分配方法。并为类增加一个用 delete 关键字释放内存的析构函数。
3. 以程序清单 2-2 为基础编写一个程序。用类继承的方法添加一个函数，该函数使用小汽车的长、宽数据计算其面积，并将计算结果在屏幕上显示出来。

第 3 学时　Qt 基础

这一学时将学习最基本的 Qt 编程知识。也许你阅读本书的目的是想学习怎样创建 GUI 程序。如果是这样的话，本学时的内容将会使你感到非常兴奋。

你将基于 QWidget 创建自己的类，这个类成为你的主窗口；它是最基本的，所有其他 Qt 对象均放置在主窗口上。

3.1　创建第一个主部件

当编写新的 Qt 应用程序时，你需要做的第一件事是创建一个主部件。你可以把主部件想象为一个工作空间，在它上面将添加按钮、滚动条、标签等等。一个程序可能有很多这样的部件组成，但它只能用一个主部件。主部件与其他 Qt 部件的区别是：当主部件终止时，整个程序就运行结束。



要确实理解什么是部件。一个部件是一个图形（并且是矩形的）对象，如一个按钮，或一个滚动条等。一个空的窗口也是一个部件。但是，当我们提到主部件时，很可能就是指主窗口，尽管还有一些其他部件。主窗口也被称作顶级部件。

创建主部件最常用的方法是基于 QWidget 或 QDialog 类创建一个用户类。本书将主要使用 QWidget 类。你可以使用户类通过公有继承派生于 QWidget 类。你应该创建一个新的构造函数，在其中调用一些成员函数（方法）来定义窗口外观。在第 2 学时“面向对象程序设计”中已经简单介绍了这一内容，这里我们将更详细地介绍。程序清单 3-1 就是这种方法的一个例子。其中创建一个空窗口，并显示在屏幕上。其大小为 200 像素乘以 120 像素。因为其最大尺寸和最小尺寸是相同的，所以不能调整窗口大小。

程序清单 3-1　　第一个主部件

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3:
4: //In the class declaration, we only need
5: //to include a new constructor. The other
6: //methods are inherited from QWidget:
```

```
 7: class MyMainWindow : public QWidget
 8: {
 9: public:
10:         MyMainWindow();
11: };
12:
13: MyMainWindow::MyMainWindow()
14: {
15:         //Set the maximum and minimum size
16:         //of the window (widget). We don't
17:         //need to use the this-pointer,
18:         //because C++ defaults to the current
19:         //class anyway:
20:         setMinimumSize( 200, 120 );
21:         setMaximumSize( 200, 120 );
22: }
23:
24: void main( int argc, char **argv )
25: {
26:         //Create the required QApplication-
27:         //object. Pass on the command-line-
28:         //arguments to the QApplication-object:
29:         QApplication a(argc, argv);
30:         //Create a MyMainWindow object, and then
31:         //set it as the main widget:
32:         MyMainWindow w;
33:         a.setMainWidget( &w );
34:         //Paint the MyMainWindow object and all
35:         //its child widgets to the screen:
36:         w.show();
37:         //Pass over control to the Qt library:
38:         a.exec();
39: }
```

在使用 g++ -lqt infile -o outfile 命令编译该源程序后，即可执行其输出文件。图 3-1 为其显示结果。

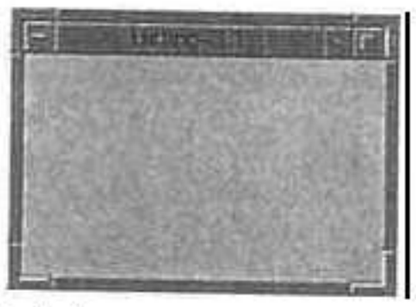


图 3-1 程序清单 3-1 所创建的 MyMainWindow 类

现在来调整窗口大小，情况怎样呢？你无法调整它！原因在于程序中的第 20、21 行。它们将窗口的最大和最小尺寸定义为同一值，因此，你不能改变窗口大小。在这两行中省略了 this 指针，如果你喜欢的话可以使用它。一些程序员认为使用它能够使程序代码变得更加容易理解。

现在来修改第 20、21 行中的值。例如，将最大值修改为 400 像素乘以 240 像素。这样你就能够将窗口大小从 200 像素乘以 120 像素调整为 400 像素乘以 240 像素。同样，如果你将最小值修改为 100 像素乘以 60 像素，那就能使窗口变得更小。

使用 QWidget::setGeometry()函数能够定义窗口大小和它在屏幕上的显示位置。这一例子如程序清单 3-2 所示。

程序清单 3-2　　一个使用 setGeometry()函数的主窗口

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3:
 4: class MyMainWindow : public QWidget
 5: {
 6: public:
 7:         MyMainWindow();
 8: };
 9:
10: MyMainWindow::MyMainWindow()
11: {
12:         setGeometry( 100, 100, 200, 120 );
13: }
14:
15: void main( int argc, char **argv )
16: {
17:         QApplication a(argc, argv);
18:         MyMainWindow w;
19:         a.setMainWidget( &w );
20:         w.show();
21:         a.exec();
22: }
```

使用 QWidget::setGeometry()函数设置窗口首次显示在屏幕上时的大小和位置。但当窗口显示后，它可以被移动和改变尺寸。程序清单 3-2 中，窗口的左上角显示在相对于屏幕左上角右边 100 像素和下面 100 像素的位置（这决定于传递给 QWidget::setGeometry()函数的前两个参数）。后两个参数表示在屏幕上所显示的窗口大小。这里，它为 200 像素宽和 120 像素高。因此，如果你希望窗口显示在距屏幕左上角右边 200 像素和下面 350 像素的位置，并且窗口为 400 像素宽和 300 像素高时，可用下面语法格式调用 setGeometry()函数：

```
setGeometry( 200, 350, 400, 300 );
```

程序清单 3-2 所显示的窗口外观如图 3-2 所示，图 3-3 为窗口移动和改变大小后的显示结果。

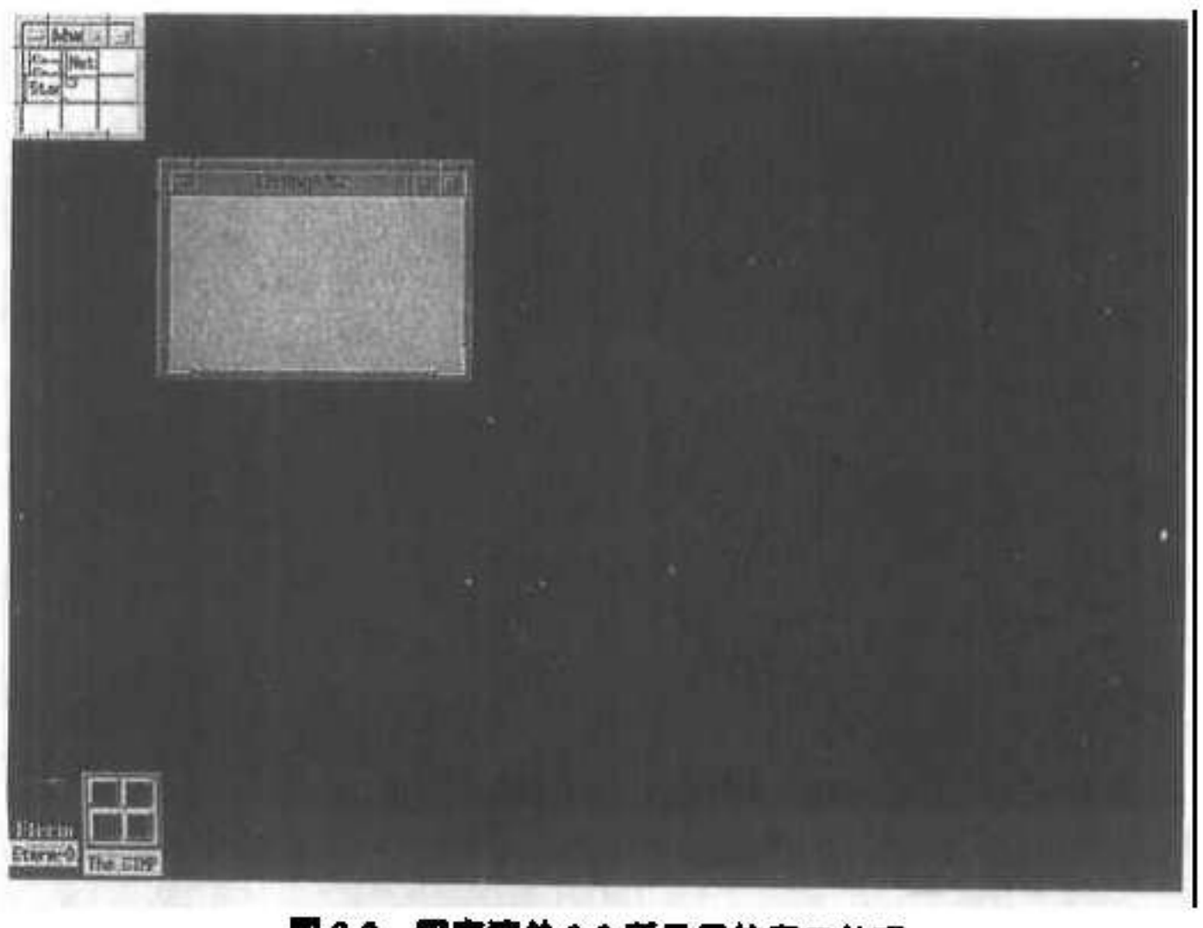


图 3-2　程序清单 3-2 所显示的窗口外观

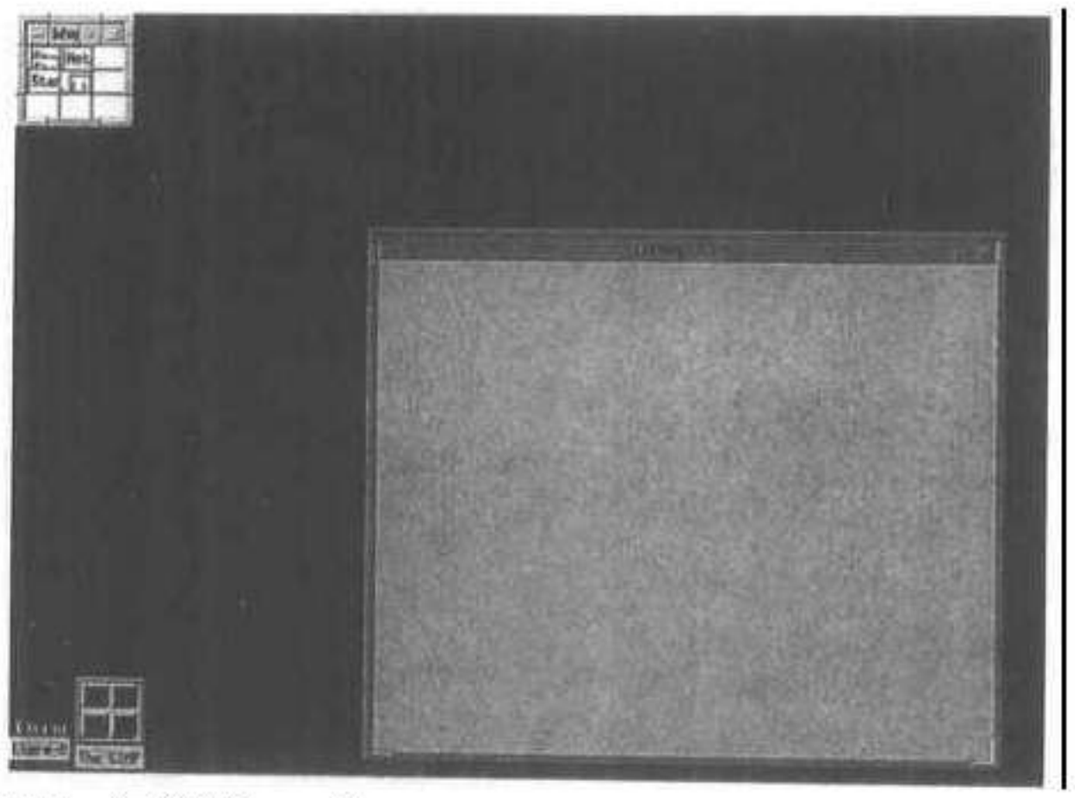


图 3-3 程序清单 3-2 所显示的窗口在被移动和改变尺寸后的显示外观

通过学习定义窗口的这两种方法，你会发现它们各有优缺点。如果在程序中不让用户调整窗口大小，那么你设计的窗口尺寸有可能不适合用户的屏幕尺寸。另一方面，如果让用户调整窗口尺寸，这将增加窗口上对象的布局难度。如果窗口没有任何子部件（对象），将不会有什么问题。

3.2 向主部件中添加对象

如果不向窗口上添加与用户交互的对象，那么你的窗口（程序）也就没有太大用处。空窗口不能做什么事情。因此，你现在应该学习怎样向刚创建的窗口上添加对象。这并不困难，只需要一些简单的 C++知识即可。例如，在添加对象时，this 指针就非常重要。实际上，没有它完全不可能实现这一操作。因此，你要真正理解什么是 this 指针，使用它能够实现哪些操作（见第 2 学时“面向对象程序设计”对 this 指针的解释）。

3.2.1 添加按钮

我们首先向窗口添加一个简单的按钮。这个按钮不做任何事情，它只不过是可见的，并且能够被用户点击。之后再向按钮添加一些格式化文本。该例子如程序清单 3-3 所示。

程序清单 3-3 具有按钮的 MyMainWindow 对象

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: //We include qpushbutton.h to be able to
 4: //use the QPushButton class:
 5: #include <qpushbutton.h>
 6: //We also include qfont.h to be able to
 7: //format the button label:
 8: #include <qfont.h>
 9:
10: class MyMainWindow : public QWidget
11: {
12: public:
```

```
13:     MyMainWindow();
14: private:
15: //Locate memory for the button:
16:     QPushButton *b1;
17: };
18:
19: MyMainWindow::MyMainWindow()
20: {
21:     setGeometry( 100, 100, 200, 120 );
22:
23:     //Here we create the button. We give two arguments
24:     //to the constructor, the fist represents the
25:     //label text, the second represents the button
26:     //parent widget. Here, the this-pointer is very
27:     //useful!
28:     b1 = new QPushButton( "Button 1", this );
29:     //Since b1 is a pointer, we use the -> operator.
30:     //We set the button's geometry by calling the
31:     //QPushButton::setGeometry() function:
32:     b1->setGeometry( 20, 20, 160, 80 );
33:     //Set the buttons label font:
34:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
35: }
36:
37: void main( int argc, char **argv )
38: {
39:     QApplication a(argc, argv);
40:     MyMainWindow w;
41:     a.setMainWidget( &w );
42:     //Since b1 is a child widget of w, both b1 and w
43:     //will be shown when we make the following call:
44:     w.show();
45:     a.exec();
46: }
```

这并不困难，不是吗？注意，第 34 行是格式化按钮上的文本。它使用 QPushButton::setFont()函数实现这一操作，函数用 QFont 对象做参数。同样也要注意传递给 QFont 构造函数的三个参数——第一个表示字体，第二个定义字体大小，第三个说明将文本加粗。最后一个参数也称作字体的粗细（Weight），它将文本定义为粗体（参看 Qt Reference Document 中的 QFont 一节，它列出了所有可用的字体粗细）。

还可以向构造函数增加第四个选项参数，它为布尔值，用于定义文本是否为斜体。看下面的例子：

```
b1->setFont( QFont("Courier", 12, QFont::Light, TRUE ) ) ;
```

这条语句将字体设置为 Courier，大小为 12 像素，文本为细体（与粗体相反）和斜体。



如果你认为下面一行

```
    b1->setFont( QFont("Times", 18, QFont::Bold) ) ;
```

看起来有点怪，可将它改为下面代码：

```
    QFont font("Times", 18, QFont::Bold) ;
    b1->setFont( font ) ;
```

这看起来更容易理解。

编译程序清单 3-3，其执行结果如图 3-4 所示。

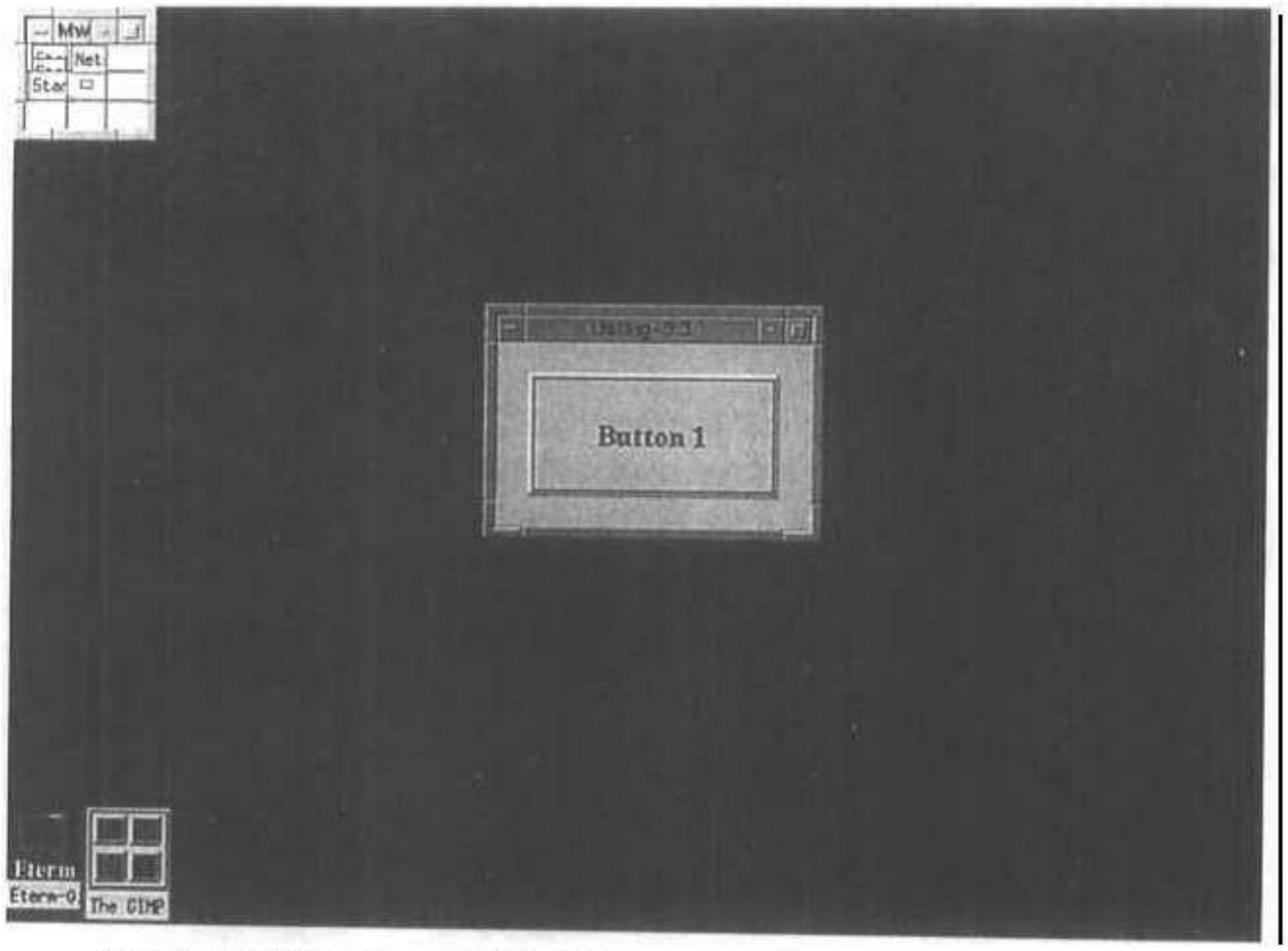


图 3-4 具有 QPushButton 对象的 MyMainWindow 类，文本标签被 QFont 格式化

3.2.2 添加标签

无论你所添加的部件是什么，向窗口中添加子部件的过程都基本相同。下面我们来看一看怎样向窗口中添加文本标签，这将用到 QLebal 类。该例子如程序清单 3-4 所示：

程序清单 3-4 具有一个按钮和一个标签的 MyMainWindow 对象

```
#include <qapplication.h>
#include <qwidget.h>
#include <qpushbutton.h>
#include <qfont.h>
//We include qlabel.h to get access to
//the QLabel class:
#include <qlabel.h>

class MyMainWindow : public QWidget
{
public:
    MyMainWindow();
private:
    QPushButton *b1;
    QLabel *label;
};

MyMainWindow::MyMainWindow()
{
    setGeometry( 100, 100, 200, 170 );

    b1 = new QPushButton( "Button 1", this );
    b1->setGeometry( 20, 20, 100, 80 );
```

```
24:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
25:
26:     //Here we create the label, define the text
27:     //and where in the window the label shall appear:
28:     label = new QLabel( this );
29:     label->setGeometry( 20, 110, 160, 50 );
30:     label->setText( "This is the first line\n
31:             This is the second line" );
32:     label->setAlignment( AlignCenter );
33: }
34:
35: void main( int argc, char **argv )
36: {
37:     QApplication a(argc, argv);
38:     MyMainWindow w;
39:     a.setMainWidget( &w );
40:     w.show();
41:     a.exec();
42: }
```

这里用到与前面添加按钮相同的方法。首先在类声明中为标签分配内存，然后在构造函数中创建和定义标签。第 28 行调用 QLebal 构造函数，并用 this 指针做参数（也就是说，它使用尚未创建的MyMainWindow对象作为标签的父窗口）。第29行用QLebal::setGeometry()函数设置标签的几何位置。在 Qt 中，文本标签的处理方法与矩形框一样，因此，可以像设置 QPushButton 对象一样设置文本标签的几何位置。第 30 行和第 31 行设置标签文本。它们包含\n 字符，该字符表示换行。第 32 行告诉 Qt 在 QLebal 矩形框内以水平方向中心对齐方式绘制文本。这个程序的执行结果如图 3-5 所示。

QLebal 类用于显示短字符串，例如，用来解释在文本框中输入什么内容等。但是，为了显示较长文本，建议不要使用 QLebal。这时，用 QMultiLineEdit 一类的部件将更适合。



图 3-5　具有一个按钮和一个标签的 MyMainWindow 类

你现在应该掌握了向窗口中添加部件的基本知识。下面我们来学习怎样处理按钮点击事件。

3.2.3　添加退出按钮

Qt 与其他同类库的区别之一在于它与用户的交互方式——也就是说，它怎样响应用户操作，例如，点击按钮或拖动滑块。其他大多 GUI 库使用所谓的回调函数方式，而 Qt 则使用信号和槽。简单地说，信号和槽是相互关联的函数。信号（Signal，实际上是一个函数）执行时，与该信号相连接的槽（Slot，实际上也是一个函数）也得到执行。这一节将给出一个这样的例子。

我们将在下一学时更加深入地讨论这一话题，这里我们来学习怎样向一个按钮添加简单的退出功能（见程序清单 3-5）。

这个清单中最令人感兴趣的部分是第 32 行。这里将 b1 按钮的 clicked()信号与 qApp 的 quit()槽相连接。当点击按钮时，将产生 QPushButton::clicked()信号，导致 qApp 的 quit()槽被执行，从而使程序退出。但是，什么是 qApp 呢？qApp 是 Qt 的一个内置指针。它总是指向程序中的 QApplication 对象（这里为 a）。其工作方式与 this 指针相同，它指向一个还没有创建的对象。它是 Qt 中一项很重要的内容，使 Qt 编程变得更加容易。这个程序的执行结果如图 3-6 所示。

程序清单 3-5　　点击按钮程序将退出

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qpushbutton.h>
 4: #include <qfont.h>
 5: #include <qlabel.h>
 6:
 7: class MyMainWindow : public QWidget
 8: {
 9: public:
10:     MyMainWindow();
11: private:
12:     QPushButton *b1;
13:     QLabel *label;
14: };
15:
16: MyMainWindow::MyMainWindow()
17: {
18:     setGeometry( 100, 100, 200, 170 );
19:
20:     b1 = new QPushButton( "Quit", this );
21:     b1->setGeometry( 20, 20, 160, 80 );
22:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
23:
24:     label = new QLabel( this );
25:     label->setGeometry( 10, 110, 180, 50 );
26:     label->setText( "If you click the button above,
27:             the whole program will exit" );
28:     label->setAlignment( AlignCenter );
29:
30:     //The following line makes the program exit when the
31:     //b1 button is clicked:
32:     connect( b1, SIGNAL( clicked() ), qApp, SLOT( quit() ) );
33: }
34:
35: void main( int argc, char **argv )
36: {
37:     QApplication a(argc, argv);
38:     MyMainWindow w;
39:     a.setMainWidget( &w );
40:     w.show();
41:     a.exec();
42: }
```



图 3-6　具有一个退出按钮和一个标签的 MyMainWindow 类

3.3　小　结

在这一学时里，你学习了怎样使用 Qt 编写简单的 OOP 程序。学习向主窗口中添加按钮和标签，实现程序的退出功能。这只用单行代码即可实现，它体现了 Qt 库优于其他 GUI 库的一个优点。只用极少量的代码就能实现这样的高级功能不仅给人留下深刻印象，而且也是一项真正的创新。

现在你熟悉了 OOP，能够使用 OOP 编写简单的 Qt 程序。在下一学时，我们将深入讨论槽和信号。

3.4　问题与答案

问：当我调整主窗口时，其中的对象位置错乱，我的程序看起来也不美观，这是为什么？

答：当调整它们的父窗口时，你应该使用特殊的方法改变子部件。这一问题我们将在第 11 学时“使用布局管理器”中讨论。

问：我不理解 setGeometry()中的参数究竟代表什么意思？

答：这一点比较容易混淆。你应该明白，像素（0, 0）表示监视器的左上角，而不是我们习惯的左下角。因此，如果你在主窗口中使用 setGeometry，setGeometry 的第一个参数定义在屏幕的左边和窗口的左上角之间的像素数。第二个参数定义在屏幕的上边和窗口的左上角之间的像素数。接下来的两个参数以像素为单位定义窗口的宽度和高度。一旦你习惯它，这也就不难理解了。

问：我的编译器显示按钮的 clicked()信号与 qApp 的 quit()槽连接这一行错误，这是为什么？

答：要确保你传递给 connect()函数的对象参数为指针（地址）。如果它们不是，你应该使用地址运算符&（将它加在对象名称前面），以获得该对象的内存地址。

3.5　作　业

完成下面的问题和练习将有助于你牢记本学时所学内容，回答这些问题有助于理解 Qt

程序设计基本知识。

3.5.1　测验

1. setMaximumSize()函数的作用是什么？
2. setMinimumSize()函数的作用是什么？
3. setGeometry()函数的作用是什么？
4. 在文件中包含 qFont.h 头文件有什么作用？
5. MyMainWindow w; 程序行的作用是什么？
6. 为什么不必调用每个对象的 show ()函数？
7. 为什么输入 this 指针代替父部件？
8. 什么是 qApp？
9. 为什么在 main ()函数中需要调用 a.exec ()函数？

3.5.2　练习

1. 基于 QWidget 编写一个新的 Qt 类，使其宽度为 400 像素，高为 300 像素。并在窗口的左上角添加一个按钮。
2. 修改上一个练习中的程序，使按钮显示在右下角（记注：必须显示出整个按钮）。
3. 将按钮移到窗口的中央。
4. 最后，将按钮的 clicked()信号与 qApp 的 quit()槽连接，并使其正常工作。
5. 在 Qt Reference Document 中查找 QPushButton 所包含的其他信号（提示：它基于 QButton）。此外，再查找 QApplication 包含哪些槽。

第 4 学时　槽和信号

Qt 部件与用户的交互方式不同于其他 GUI 工具包。用户交互是所有 GUI 应用程序关心的问题。通过将某种用户事件（如点击鼠标）与程序事件（例如程序退出）联系起来，使用户能够在图形界面中只使用鼠标来控制程序。

其他工具包使用回调函数创建用户交互。但是，回调函数非常复杂，容易混淆，又难以理解（至少大部分 Qt 程序员这样认为）。因此，Qt 开发者使用另一种方法来完成这一工作。这种方法依赖于 Qt 特有的两个功能：信号和槽。使用这种新方法只需要一行代码就能够将用户事件和程序事件连接起来。

将用户事件连接到程序事件这种方法要比回调函数更加容易使用。槽和信号是你选择 Qt 库而不是其他工具包的两个重要原因。

在这一学时，你将学习信号和槽的基本知识和 Qt 预定义的一些信号和槽。在这一学时的最后部分，你将学习怎样定义你自己的信号和槽。这需要使用元对象编译器（Meta Object Compiler，MOC）实用程序，这一学时也将介绍该内容。

4.1　理解信号和槽

首先，你必须理解信号（signal）和槽（slot）。这一技术有点不同于传统的回调（call back）函数。信号和槽技术是由 Troll Tech 公司独立开发的——而不是 C++的功能。为了完全理解本学时所讨论的信号和槽，你应该理解几个术语。这些术语如表 4-1 所示。

表 4-1　　槽和信号术语

术　语	解　释
用户事件	指程序的用户所产生的事件。例如，点击鼠标或拖动滚动条
程序事件	指程序所产生的事件。例如，当用户点击鼠标后程序退出
发射信号	基本意思是指“发出”一个信号。例如，当你点击鼠标时，将发射 clicked() 信号。为了发射信号，使用 emit 关键字
内部状态	当谈论信号时，你可能听说过“内部状态已经改变”。这意味着对象内部数据已发生改变，因此对象发射一个信号
MOC	元对象编译器用于构造用户自己的信号和槽。它处理 Qt 特有的关键字（例如 emit），创建合法的 C++ 代码。本学时后面的“元对象编译器”一节将更多介绍 MOC

4.1.1　槽

实际上，槽是标准的成员函数（作为类成员的函数）。但是，它们增加了一些特殊的功

能使它们能够连接到信号。每当槽所连接的信号被发射时，槽（函数）就被执行。因此，当你在第 8 学时“认识 Qt 部件：第 3 课”中创建自己的槽时，实际上是在编写一个普通的成员函数。

许多 Qt 类已经包含一些程序中可以使用的预定义槽。QPushButton 类所包含的 clicked() 函数就是一个很好的例子。

槽实际上是标准函数，因此，你可以像调用其他函数一样调用它们。

4.1.2 信号

信号也是成员函数，但是，它们的实现有一点不同（这将在本学时后面的“元对象编译器”一节中介绍）。

当对象内部发生某些事件（也就是其内部状态发生改变）时，它能够发出信号（但它不一定必须这样）。如果这个信号连接到槽，那么那个槽（函数）就被执行。可以将多个槽连接到同一个信号，那么这些槽就将一个接一个地被执行，其执行顺序是任意的。

因此，信号是一种特殊类型的函数。它们被定义为当某个事件发生时就被发射。之后，执行所有被连接的槽。

4.2 使用预定义信号和槽

在上一学时“Qt 基础”一章的程序中实现退出按钮功能时，使用了信号和槽。我们将按钮的 clicked()信号连接到 qApp 的 quit()槽。这使得每当发射 clicked()信号时，程序便执行 quit()槽（函数）。

当然，Qt 还包含许多其他信号和槽，你可以以各种方法来使用它们。这一节介绍预定义信号和槽，以及它们的一些使用方法。

4.2.1 例 1——QSlider 和 QLCDNumber

槽和信号能够用来创建许多比退出按钮更令人激动的对象。尽管下一个例子不是很奇特，但它肯定比上一学时中的退出按钮更令人感兴趣。程序清单 4-1 向你介绍怎样使用槽和信号：

程序清单 4-1 槽/信号例子

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qpushbutton.h>
4: #include <qfont.h>
5: #include <qlcdnumber.h>
6: #include <qslider.h>
7:
8: class MyMainWindow : public QWidget
9: {
10: public:
11:    MyMainWindow();
12: private:
13:     QPushButton *b1;
```

```
14:     QLCDNumber *lcd;
15:     QSlider *slider;
16: };
17:
18: MyMainWindow::MyMainWindow()
19: {
20:     setGeometry( 100, 100, 300, 200 );
21:
22:     b1 = new QPushButton( "Quit", this );
23:     b1->setGeometry( 10, 10, 80, 40 );
24:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
25:
26:     lcd = new QLCDNumber( 2, this );
27:     lcd->setGeometry( 100, 10, 190, 180 );
28:
29:     slider = new QSlider( Vertical, this );
30:     slider->setGeometry( 10, 60, 80, 130 );
31:
32:     //The following line makes the program exit when the
33:     //b1 button is clicked:
34:     connect( b1, SIGNAL( clicked() ), qApp, SLOT( quit() ) );
35:
36:     //The following line connects the slider to the LCD display,
37:     //and makes the display number change as you drag the slider.
38:     connect( slider, SIGNAL( valueChanged(int) ),
39:              lcd, SLOT( display(int) ) );
40: }
41:
42: void main( int argc, char **argv )
43: {
44:     QApplication a(argc, argv);
45:     MyMainWindow w;
46:     a.setMainWidget( &w );
47:     w.show();
48:     a.exec();
49: }
```

在程序清单 4-1 中，connect()函数用于连接 b1 的 clicked()信号和 qApp 的 quit()槽（第 24 行）。其结果导致当点击 b1 时，程序退出。在第 38 行，slider 的 valueChanged()信号被连接到 lcd 的 display()槽。注意，一个 int 值在两个函数之间传递。这使得通过拖动滑块能够控制显示中所显示的数值。



当在类定义中调用函数而没有指定函数所属类时，C++认为它是当前类（你所定义类）。因此，前面例子中调用 connect()函数而没有指定其所属类是正确的。

编译并执行上面例子，当你拖动滑块时，观察一下会发生什么。它是不是很有趣？其执行结果如图 4-1 所示。

与你想象的一样，槽和信号能够实现除退出按钮以外的许多其他功能。下一节将给出另外一个例子。

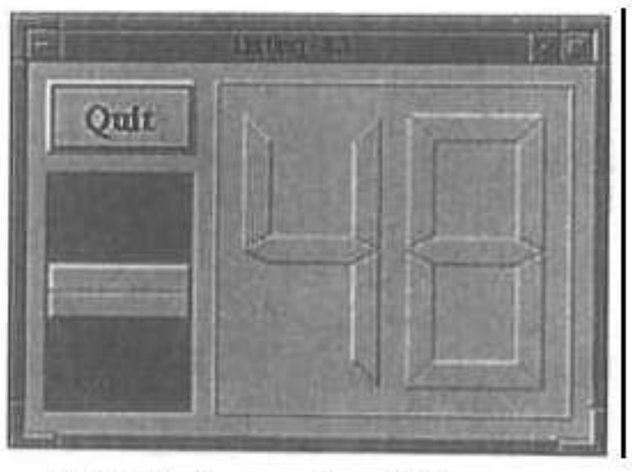


图 4-1　具有滑块和 LCD 显示的槽和信号应用实例

4.2.2　例 2——QPushButton 和 QLineEdit

现在，你将用三个 QpushButton 对象控制是否选定、取消选定或删除一个 QLineEdit 对象中的文本。这通过将 QPushButton 的 clicked()信号连接到 QLineEdit 对象的适当槽来实现。

程序清单 4-2 所示的这个例子虽不能够引人入胜，但它会给你一些启示：

程序清单 4-2　　连接三个 QPushButton 对象和一个 QLineEdit

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qpushbutton.h>
 4: #include <qfont.h>
 5: #include <qlineedit.h>
 6: #include <qstring.h>
 7:
 8: class MyMainWindow : public QWidget
 9: {
10: public:
11:     MyMainWindow();
12: private:
13:     QPushButton *b1;
14:     QPushButton *b2;
15:     QPushButton *b3;
16:     QLineEdit *ledit;
17: };
18:
19: MyMainWindow::MyMainWindow()
20: {
21:     setGeometry( 100, 100, 300, 200 );
22:
23:     b1 = new QPushButton( "Click here to mark the text", this );
24:     b1->setGeometry( 10, 10, 280, 40 );
25:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
26:
27:     b2 = new QPushButton( "Click here to unmark the text", this );
28:     b2->setGeometry( 10, 60, 280, 40 );
29:     b2->setFont( QFont( "Times", 18, QFont::Bold ) );
30:
31:     b3 = new QPushButton( "Click here to remove the text", this );
32:     b3->setGeometry( 10, 110, 280, 40 );
33:     b3->setFont( QFont( "Times", 18, QFont::Bold ) );
34:
```

```
35:     ledit = new QLineEdit( "This is a line of text", this );
36:     ledit->setGeometry( 10, 160, 280, 30 );
37:
38:     //The following three lines connects each button to a predefined
39:     //slot on the ledit object. Test the program to find out what
40:     //happens!
41:     connect( b1, SIGNAL( clicked() ), ledit, SLOT( selectAll() ) );
42:     connect( b2, SIGNAL( clicked() ), ledit, SLOT( deselect() ) );
43:     connect( b3, SIGNAL( clicked() ), ledit, SLOT( clear() ) );
44: }
45:
46: void main( int argc, char **argv )
47: {
48:     QApplication a(argc, argv);
49:     MyMainWindow w;
50:     a.setMainWidget( &w );
51:     w.show();
52:     a.exec();
53: }
```

该程序执行结果如图 4-2 所示。

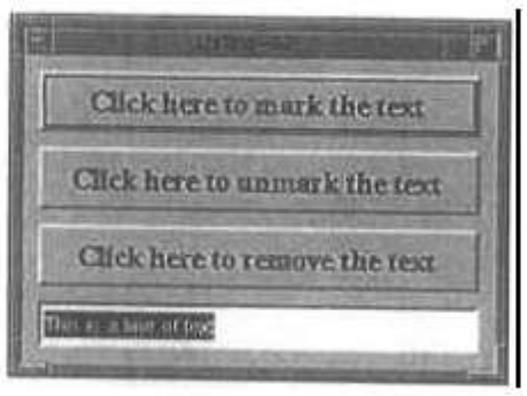


图 4-2　怎样使用 QLineEdit 对象中的槽和 3 个 QPushButton 对象

这个例子使用 QPushButton 的 clicked()信号（一个非常有用的信号），并将它连接到 QLineEdit 类所包含的 3 个不同槽（第 41、42、43 行）。这样做，点击这些按钮就能够控制 QLineEdit 对象中的文本。

我还能够编写许多类似于这一学时中你所看到的程序实例，但是，这两个例子已经足够使你理解信号和槽的一般用法。如果需要研究更多的程序实例，你可以阅读 Qt Reference Document，其中包含所有 Qt 信号和槽信息。

4.3　创建和使用用户信号和槽

当编写 Qt 应用程序时，你会花费很多时间用于创建用户槽（换句话说，也就是你自己定义的槽）。当然，你也可以创建你自己的信号。但是，由于在大多情况下预定义信号已经足够用，所以，不常这样做。这一节你将学习怎样创建用户槽和信号，和怎样使用元对象编译器，创建用户槽和信号时需要使用它。

4.3.1　认识元对象编译器

如前所述，信号/槽技术非常特殊。实际上，需要特殊的编程语句才能创建信号和槽。

因此，C++编译器不能理解这些语句，因此，完成这项工作必须使用特殊的工具，即元对象编译器（Meta Object Compiler，MOC）。这一工具扫描源文件中用于创建信号和槽的特殊语句，并将它生成为有效的 C++代码（编译器能够识别的代码）。这一节将介绍怎样使用 MOC。

4.3.2　定位元对象编译器

元对象编译器二进制文件 moc 总是存放在$QTDIR/bin 目录下。因此，如果安装 Qt 的基本目录为/usr/local/qt，则所有的 Qt 二进制文件，包括 moc，都将存放在/usr/local/qt/bin 目录中。当编译 Qt 库时，二进制文件被自动编译。

如果按照第一学时“Qt 简介”中介绍的方法，将$QTDIR/bin 目录添加到 PATH 变量中，则在文件系统中的任何位置只需要在提示符后输入 moc 便可执行 moc。

使用元对象编译器

当在类声明中定义完信号和槽后（本学时后面部分将介绍），你需要使用 moc 来读取文件，并从中提取有效的 C++代码。moc 的用法非常简单，其语句为：

```
$ moc infile.h -o outfile.moc
```

这个命令读取 infile.h 文件，该文件中声明 Qt 类，以及信号和/或槽，然后根据它创建有效的 C++代码，并将这些代码写入 outfile.moc 文件。非常简单！

为了编译新类，你可以在 cpp 文件（类定义文件）中包含 moc 文件（moc 创建的文件），或者从 moc 文件和 cpp 文件创建目标文件，然后将它们链接到一起。例如：

```
$ g++ -c outfile.moc -o outfile.o
$ g++ -c infile.cpp -o infile.o
$ g++ -lqt outfile.o infile.o -o MyProgram
```

这两种方法（包含 moc 文件或一起链接）的效果完全相同，你可以选择最适合你的方法。本书将使用前一种方法。

4.4　创建用户槽

如果你想产生与程序用户操作（如点击按钮）相对应用户事件，则必须创建能够连接到适当信号（如使用按钮时的 clicked()信号）的用户槽。为此，你需要了解以下两项内容：怎样使用 moc 和怎样在类声明中定义槽。读完本节后，你应该知道怎样使用 moc。但是，你还不知道怎样在类声明中定义槽，你将在下一节中学到这一内容。

4.4.1　声明用户槽

创建用户槽所需做的第一件事情就是在类声明中声明它。一个槽实际上是一个普通的成员函数，但是，你需要使用特殊关键字，使 moc 能够将槽和其他成员函数区分开来。看下面的类声明：

```
class MyMainWindow : public QWidget
{
public:
    MyMainWindow ( );
```

```
};
```

这个类只有一个构造函数，在其中你可以创建和使用预定义槽和信号来连接部件和事件。但是，如果你要实现预定义槽未实现的功能，则需要定义用户槽。例如，如果 QApplication::quit()槽不存在，你则需要自己编写能够使程序退出的槽。首先，需要在类声明中添加下面槽：

```
class MyMainWindow : public QWidget
{
    Q_OBJECT
public:
    MyMainWindow ( );
public slots:
    MyExitSlot ( );
};
```

你感到不熟悉的第一件事可能是对 Q_OBJECT 宏的调用。所有具有用户信号和槽的 Qt 类必须用到这个宏，否则，它将无法工作。你不必了解为什么必须这样做，只要记住在类声明中包含它即可。

像你所看到的，这里使用一种新的语法所定义的新函数叫作槽。当你在这个声明中使用 moc 时，它会发现下面一行：

```
public slots:
```

并将这行后的所有函数看作槽（直到类声明结束或是另一部分开始，如 private:）。如果你想在类中声明多个槽，只需在 MyExitSlot ()后一个个列出它们即可。

4.4.2　定义用户槽

当在类声明中声明槽之后，你需要编写实际槽定义（函数）。这像在 cpp 文件中定义其他函数一样。例如，对于 MyExitSlot ()槽，cpp 文件为：

```
#include <stdlib.h>
#include <qpushbutton.h>

//The slot:
void MyMainWindow::MyExitSlot ( )
{
    //use the exit( ) function defined in stdlib.h
    exit(0);
}

//The constructor
MyMainWindow:: MyMainWindow( )
{
    //some code
    //some code
```

```
    //connect a button to our custom slot:
    connect ( MyButton, SIGNAL( clicked( ) ), this, SLOT( MyExitSlot() ) );
}
```

这里令人感兴趣的部分是槽的定义。正如你所看到的，它是一个非常简单（而且很有用）的槽，它使用 stdlib.h 中定义的 exit()函数使程序退出。如果你在编写一个完整的程序（在构造函数中使用真正内容来创建和放置 MyButton 按钮），就可以采用上一个例子中的方法来连接它，其执行结果应与程序清单 3-5（在第 3 学时“Qt 基础”中）相同。

4.4.3 编译使用用户槽程序

如前所述，当程序中所使用的类具有用户槽和信号时，你需要使用 moc。使用 MyExitSlot() 槽的例子也需要使用 moc。首先，对类声明使用 moc：

```
$ moc ClassDeclaration.h -o ClassDeclaration.moc
```

然后，在 cpp 类文件（该文件具有函数定义）中包含 moc 文件：

```
#include < ClassDeclaration.moc >
```

最后，编译 cpp 文件，并将它与 Qt 库链接（当然，为了使程序正常运行，你需要实现 main()函数）：

```
$ g++ -lqt ClassDeclaration.cpp main.cpp -o MyProgram
```

此外，你也可以将 moc 文件编译为目标文件，之后把它与 ClassDefinifion.cpp 和 main.cpp 一起链接，其效果与前者相同。下面是其操作步骤：

```
$ moc ClassDeclaration.h -o ClassDeclaration.moc
$ g++ -c ClassDeclaration.moc -o ClassDeclaration.o
$ g++ -c ClassDefinition.cpp -o ClassDefinition.o
$ g++ -c main.cpp -o main.o
$ g++ -lqt ClassDeclaration.o ClassDefinition.o main.o -o MyProgram
```

在这种情况下，不要在 ClassDefinition.cpp 文件中包含 ClassDeclaration.moc 文件。

你已经声明、定义和使用了用户槽。尽管这是一个非常简单的例子，但是无论槽定义有多长或多么复杂，与这里所解释的概念是相同的，在本书后面部分你将学习更多的用户槽和怎样定义、使用它们的例子。

4.4.4 创建用户信号

你可能偶尔会需要定义自己的信号，但不会经常需要这样做（肯定不会像需要定义用户槽那样频繁）。因此，你可以粗略地浏览这一节。

首先，你需要在类声明中声明用户信号。如同在 public slots:下声明槽一样，信号应该在 signals:关键字下声明。因此，具有用户信号的类声明格式应该像下面这样：

```
class MyMainWindow : public QWidget
{
    Q_OBJECT
public:
    MyMainWindow( );
```

```
        SetValue( int );
    public slots:
        ChangeValue( int );
    signals:
        ValueChanged( int );
    };
```

这里，声明一个用户信号，MyCustomsignal ()。该信号带一个 int 参数。此外，也声明一个用户槽 ChangeValue()，它也带一个 int 参数。这里还声明一个普通函数，SetValue()。

正如你可能猜想到的，只有当一个新的值传递给 SetValue()函数时，SetValue()函数才应该调用 ValueChanged()信号。之后，通过将 ValueChanged()信号连接到 ChangeValue()槽，使得当有新值传递给 SetValue()函数时，能够引起数值的变化。多数情况下，这是不必要的，但它演示了信号的使用方法。SetValue()函数可以下类似格式实现：

```
void MyMainWindow::SetValue ( int value)
{
    if ( value != oldvalue )
    {
        oldvalue = value;
        emit ValueChanged( value );
    }
}
```

如你看到的，只有当新值与旧值不同时才发射 ValueChanged()信号。如果这样，oldvalue 将被修改为 value，并发射 ValueChanged()信号。应注意的是，信号与槽一类的普通函数不同，它只能使用 emit 关键字发射。ChangeValue()可定义为：

```
void MyMainWindow::ChangeValue( int value )
{
    FunctionForChangingTheValue( value );
}
```

在这段代码中，调用 FunctionForChangingTheValue()函数去修改数据。你需要做的最后一件事是将信号和槽连接起来：

```
connect( this, SIGNAL( ValueChanged( int ), this, SLOT( ChangeValue( int ) ) );
```

这个例子的功能是当调用 SetValue()函数时，它检查新值是否等于旧值。如果不等，则发射 ValueChanged()信号。因为 ValueChanged()信号被连接到 ChangeValue()槽，因此当发射 ValueChanged()信号时将执行 ChangeValue()槽。之后，ChangeValue()槽调用 FunctionForChangingTheValue()函数去修改实际数据。

是不是对此感到有点混乱？的确是这样。但是，这些是你编写比较复杂的 Qt 应用程序（和所有其他 GUI 应用程序）必须面对的问题。你所编写的代码应尽可能流畅而且使人容易理解。你所看到的这个例子并不是修改 int 值的最佳实现方法，但它显示了怎样使用信号。这是整个问题的关键。应该记注：当类声明中包含用户信号或槽时要对它使用 moc。

在阅读本节后，你应该理解怎样创建和使用用户信号和槽。如前所述，在本书后面部分你将看到更多关于怎样使用用户槽的例子，当学习其他学时时，你将学到更多这方面的

知识。

4.5 信号和槽的有趣功能

现在你已经基本了解了信号和槽的用法。但是，你会发现信号和槽还有一些其他有趣功能非常有用，例如，断开槽和信号之间的连接，连接信号和信号。这一节将介绍这些功能和其他几个功能。

4.5.1 避免不必要的信息

在程序清单 4-1 中使用下面代码将滑块连接到 LCD 显示：

```
connect ( slider, SIGNAL( valueChanged( int ) ),
    lcd, SLOT( display( int ) ) ) ;
```

注意，这里 valueChanged()信号和 display()槽均使用 int（整数）值做参数。这意味着信号将注意滑块的移动，并告诉移动后的新值。这个 int 值被发送到所连接的 display()槽，处理后被显示到 LCD 上。

但是，有时一个信号所给出的信息比其已连接槽所需要的信息要多。例如，你可能只关心一个触发按钮的触发时间，而不需要知道它的触发状态。在这种情况下，你可以像下面这样在槽定义中省略相应的参数（注意：接收对象及其槽是虚构的，它们并不存在）：

```
connect( togglebutton, SIGNAL( toggled(bool) ),
   anotherobject, SLOT( aslot( ) ) ) ;
```

当你连接信号和槽时，如果它们不真正匹配时，可以使用这种方法。

4.5.2 信号和信号之间的连接

有时，你可能需要将一个信号连接到其他信号。尽管这听起来有点离奇，但完全有可能需要这样做。例如，你可能想在点击一个按钮时产生其他某种事件——很容易通过调用其他信号启动的事件，一个很好的想法是将按钮的 clicked()信号连接到那个事件的启动信号。

当你想声明一个连接其他信号的信号时，只需像下面这样修改 connect()函数的第 3 个参数即可：

```
connect( button, SIGNAL( clicked( ) ), this, SIGNAL( anothersignal( ) ) ) ;
```

现在，每当发射按钮的 clicked()信号时，就好像同时发射当前定义类的 anothersignal()信号一样。之后，如果你将 anothersignal()连接到某个槽，则每当发射 clicked()信号时，那个槽（函数）将被执行。

4.5.3 断开槽和信号之间的连接

有时，你可能需要断开信号和槽（或信号和信号）之间的连接。以在程序里暂时禁止一个函数的执行。使用 QObject::disconnect()函数就能够实现这样的功能，它所需的参数与 QObject::connect()函数完全一样。例如，如果需要断开 clicked()信号和 quit()槽之间的连接，使用下面的代码即可：

```
disconnect( button, SIGNAL( clicked( ) ), qApp, SLOT( quit( ) ) ) ;
```

如果在包含退出按钮的例子中添加这行代码，将导致用户再点击该按钮时无法结束程序的执行。

4.5.4　使用 connect()函数时省略对象名称

connect()函数的这个功能能够减少击键次数。如果将信号连接到当前所定义类（在该类方法里）中的槽时，你可以省略拥有该槽的对象名称。例如，我们来看下面一个基于 QWidget 类的方法：

```
MyWidget::MyWidget( )
{
    //some code here (including the
    //definition of button):
    ...
    ..
    .
    //and then the call to connect( ):
    connect( button, SIGNAL( clicked( ) ), SLOT( ckearFocus( ) ) ) ;
}
```

在这个例子中，省略第 3 个参数。connect()函数默认为当前所定义对象。尽管这种方法能够减少击键次数，但并不建议你使用它，因为，它使程序代码变得难以理解。尽管不推荐使用这种方法，但你应该了解这种方法，因为其他人可能会使他们的程序代码使用这种方法。

4.6　小　结

在这一学时里，你学习了怎样使应用程序与用户进行交互。事实上，这是 GUI 程序设计的全部思想，因此，为了能够创建好的程序，你需要真正理解这一内容。

Qt 中信号和槽的实现非常逼真，因此，你学习和理解这一学时中的例子不会有任何问题。信号和槽也容易使用，它使你的程序更加容易理解并且不容易出错。

但是，为了构造真正的 GUI 应用程序，你需要了解 Qt 构造块知识。这些将在下一学时“学习 Qt 构造块”中介绍。

4.7　问题与答案

问：我的编译器报告未找到 connect()函数，这是为什么？

答：如果调用 connect()函数而未定义从哪个类中查找它，则将出现这种错误。这种调用必须在一个 QObject 派生类的方法内。如果从一个外部函数中调用，你必须像下面这样调用：

```
QObject::connect( );
```

问：我的编译器报告传递给 connect()函数的参数错误，这是为什么？

答：connect()函数参数使用对象指针，而不是对象自身。因此，你必须确认所提供的参数是正确的。

4.8 作　业

希望你完成下面的问题和练习，这有助于你牢记本学时所学内容。完成这些问题和练习，你能够理解信号和槽的工作方式，以及在程序中怎样实现它们。

4.8.1 测验

1. 什么是槽？
2. 什么是信号？
3. 怎样将信号连接到槽？
4. 能否将多个槽连接到一个信号？
5. 什么时候能够调用 connect()函数而不指定定义它的类？
6. 能够将被连接的槽和信号断开吗？
7. 在调用 connect()函数时，省略槽所属对象名称意味着什么？
8. 是否能够将一个信号连接到其他信号？如果能，应该怎样操作？

4.8.2 练习

1. 基于程序清单 4-2 编写一个程序，增加一个用于修改 QLineEdit 对象文本的按钮（提示：请使用 Qt Reference Document）。
2. 在程序清单 4-2 中使用 disconnect()函数断开所有连接，然后重新编译该程序，看看会发生什么？
3. 编写一个具有两个 QPushButton 对象和一个单选按钮（查阅 Qt Reference Document，寻找用于单选按钮的 Qt 类）的程序。连接这 3 个对象，使两个 QPushButton 对象能够控制单选按钮是否被选取——一个用于选取，另一个取消选取。

第 5 学时　深入学习 Qt 构造块

这一学时，你将看到 Qt 为创建 GUI 应用程序所提供的更多内容。将学习怎样使用滚动条、菜单和文件 I/O，也将学习 QmainWindow 部件的用法。

你在这一学时所学习的知识是 Qt 程序设计所必须的内容。你会更好理解 Qt 所提供的内容，以及在程序中怎样使用和实现 Qt 类。实际上，所有的 Qt 部件均以十分标准的方法实现。因此，当你学习使用几个部件之后，对其他部件的使用就能做到融会贯通。在学习新部件时，Qt Reference Document 将是你的一个好助手。

在这一学时中，首先学习向程序中添加滚动条。这是现代 GUI 应用程序能够在较小的区域内放置更多部件所使用的方法。之后，将继续学习 Qt 的菜单创建方法。在现代 GUI 应用程序中使用菜单已成为一种标准。它们用于向用户显示程序功能和选项，并使组织这些功能和选项工作变得更加简单。本学时最后将讨论 Qt 的 QMainWindow 类，它使大家能够更容易地创建标准程序。

如果你渴望学习创建真正的 GUI 程序，这一学时将会令你非常激动。

5.1　使用滚动条

滚动条使你能够在屏幕上创建显示区域非常大的应用程序。在很多情况下，应用程序需要滚动条才能在桌面上运行。使用滚动条，能够使用户更加容易地使用应用程序（因为无论他们桌面的分辨率设置是多少，他们都能够看到所有的部件）。例如，如果你所创建的应用程序适合在 1024×768 像素的桌面上运行，那么，那些使用 800×600 或 640×480 像素桌面的用户将无法看到程序的整个外观（因为他们的桌面不够大）。因此，学习这一节内容，使你的程序永远不要遭遇这一问题。

5.1.1　了解滚动条

使用 Qt 时，实现滚动条功能最简单的方法是将主部件（主窗口）基于 QScrollView 类。之后再为每个子窗口调用 QScrollView::addChild()函数，就能够自动创建一个按需滚动窗口（即只有当窗口无法在其原来尺寸下显示出所有子部件时它才添加滚动条）。但是，当不需要滚动条时，将不会显示滚动条。下面程序清单 5-1 就是这样一个例子：

程序清单 5-1　　一个 QScrollView 例子

```
1: #include <qapplication.h>
2: #include <qpushbutton.h>
```

```
 3: #include <qfont.h>
 4: #include <qscrollview.h>
 5:
 6: //We base our new class on QScrollView
 7: //instead of QWidget:
 8: class MyMainWindow : public QScrollView
 9: {
10: public:
11:     MyMainWindow();
12: private:
13:     QPushButton *b1;
14: };
15:
16: MyMainWindow::MyMainWindow()
17: {
18:     //Set the geometry for the scrollview:
19:     setGeometry( 100, 100, 200, 100 );
20:
21:     b1 = new QPushButton( "This button is not too\n big for the window!",
this );
22:     b1->setGeometry( 10, 10, 180, 80 );
23:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
24:
25:     //We must call the QScrollView::addChild function
26:     //for each of its child widgets:
27:     addChild( b1 );
28: }
29:
30: void main( int argc, char **argv )
31: {
32:     QApplication a(argc, argv);
33:     MyMainWindow w;
34:     a.setMainWidget( &w );
35:     w.show();
36:     a.exec();
37:}
```

注意，程序中第 8 行所声明的类是基于 QScrollView，而不是 QWidget。此外，也要注意第 27 行中对 QScrollView::addChild()函数的调用。这一调用告诉 QscrollView 应该考虑哪个窗口需要滚动条。当基于 QScrollView 的窗口不能放置下 QScrollView::addChild()函数所添加的子部件时，它添加垂直或水平滚动条（或者同时添加二者），使用户使用它们能够看到部分或者全部放置在该窗口外的子部件。将窗口基于 QScrollView，并使用 QScrollView::addChild()函数，能够向所有需要滚动条的窗口添加滚动条。程序清单 5-1 所示例子的执行结果如图 5-1 所示，图示情况为其原始尺寸（即第一次出现在屏幕上时的状态）。

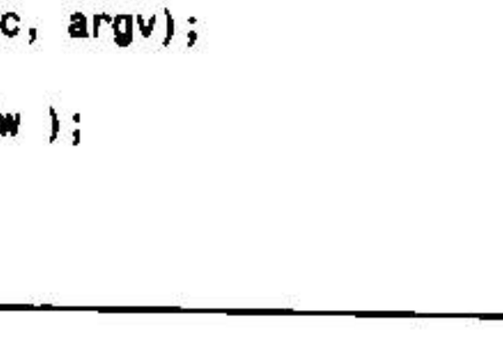


图 5-1 一个基于 QScrollView 的简单例子，这里不需要滚动条

但是，这里不需要滚动视图功能。为了演示该功能的使用，你必须调整窗口大小（用

鼠标拖动窗口的一角），使它不能显示出所有子部件（这里只有一个 QPushButton 对象）。首先缩小窗口的宽度，使它显示出图 5-2 所示的显示结果。



图 5-2　当窗口宽度缩小到不能显示出整个按钮时水平滚动条就会出现

正如你所看到的，出现了一个滚动条，这时你可用鼠标拖动它去观察被隐藏的部分按钮。相反，如果缩小窗口高度，你会看到类似于图 5-3 所示的显示结果。

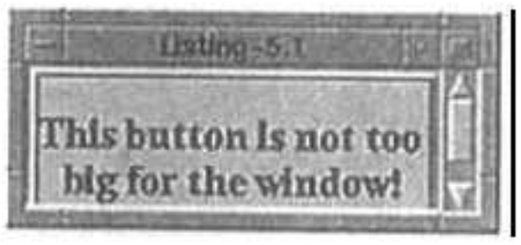


图 5-3　当窗口高度缩小到不能显示出整个按钮时垂直滚动条就会出现

这次，出现了垂直滚动条。当缩小窗口高度时，使用它能够看到按钮被隐藏的部分。最后，你可以同时缩小窗口宽度和高度，这样可以使水平滚动条和垂直滚动条同时出现。如图 5-4 所示。

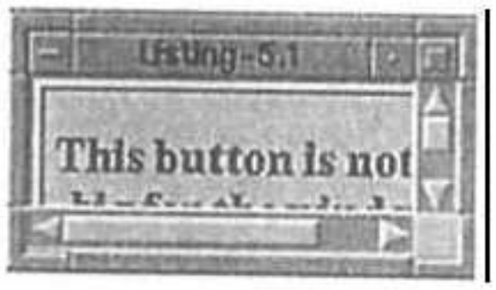


图 5-4　当窗口宽度和高度都缩小到不能显示出整个按钮时水平和垂直滚动条就会同时出现

刚才我们所介绍的就叫作按需滚动——即不需要滚动条时，它们就不被显示出来。但是，你也可以使用 QScrollBar 类手工添加滚动条（我不推荐你这样做，因为这些是不必要的工作——实际上这些完全可以由 QScrollView 类自动实现）。因此，你应该坚持使用 QScrollView 类，它更容易实现。

5.1.2　一个实际的例子

程序清单 5-2 给出一个滚动条例子。这个例子使用几个按钮和一个文本区域创建一个简单的版面。这些部件均是 QWidget 对象的子对象。使用 QScrollView::addChild()函数将 QWidget 对象添加到滚动视图，从而使它能够在必要时向窗口添加滚动条。

程序清单 5-2　　一个使用 QScrollView 的真实例子

```
1: #include <qapplication.h>
2: #include <qpushbutton.h>
3: #include <qfont.h>
4: #include <qmultilineedit.h>
5: #include <qscrollview.h>
6: #include <qwidget.h>
```

```
 7:
 8: //We base our new class on QScrollView
 9: //instead of QWidget:
10: class MyMainWindow : public QScrollView
11: {
12: public:
13:     MyMainWindow();
14: private:
15:     //We create a QWidget object as a child widget
16:     //to the QScrollView object instead:
17:     QWidget *main;
18:     QPushButton *b1;
19:     QPushButton *b2;
20:     QPushButton *b3;
21:     QPushButton *b4;
22:     QPushButton *b5;
23:     QMultiLineEdit *edit1;
24: };
25:
26: MyMainWindow::MyMainWindow()
27: {
28:     //Set the geometry for the scrollview:
29:     setGeometry( 100, 100, 470, 410 );
30:
31:     main = new QWidget( this );
32:     main->resize( 460, 400 );
33:
34:     b1 = new QPushButton( "New", main );
35:     b1->setGeometry( 10, 10, 80, 30 );
36:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
37:
38:     b2 = new QPushButton( "Open", main );
39:     b2->setGeometry( 100, 10, 80, 30 );
40:     b2->setFont( QFont( "Times", 18, QFont::Bold ) );
41:
42:     b3 = new QPushButton( "Save", main );
43:     b3->setGeometry( 190, 10, 80, 30 );
44:     b3->setFont( QFont( "Times", 18, QFont::Bold ) );
45:
46:     b4 = new QPushButton( "Print", main );
47:     b4->setGeometry( 280, 10, 80, 30 );
48:     b4->setFont( QFont( "Times", 18, QFont::Bold ) );
49:
50:     b5 = new QPushButton( "Quit", main );
51:     b5->setGeometry( 370, 10, 80, 30 );
52:     b5->setFont( QFont( "Times", 18, QFont::Bold ) );
53:
54:     edit1 = new QMultiLineEdit( main );
55:     edit1->setGeometry( 0, 50, 440, 340 );
56:     edit1->setText( "Let's pretend this is a text editor." );
57:
58:     //We must call the QScrollView::addChild function
59:     //for each of its child widgets. Since main holds
60:     //all our other widgets, we only have to insert that
61:     //one:
62:     addChild( main );
63: }
64:
```

```
65: void main( int argc, char **argv )
66: {
67:     QApplication a(argc, argv);
68:     MyMainWindow w;
69:     a.setMainWidget( &w );
70:     w.show();
71:     a.exec();
72: }
```

这个程序清单显示出一个使用 QWidget 对象的新方法：创建 QWidget 对象，使它作为 QScrollView 派生类的子类。在前面例子中，QWidget 类只是被用做基类，但是，这个例子说明 QWidget 同样也可用做子部件。之后，创建按钮和 QMultiLineEdit 对象，它们作为 QWidget 对象的子部件，这样做，你只需调用一次 QScrollView::addChild()（添加 QWidget 对象）。如果这里不使用 QWidget 对象，你需要为每个按钮和 QMultiLineEdit 对象调用 QScrollView::addChild()函数。这个程序运行结果如图 5-5 所示。

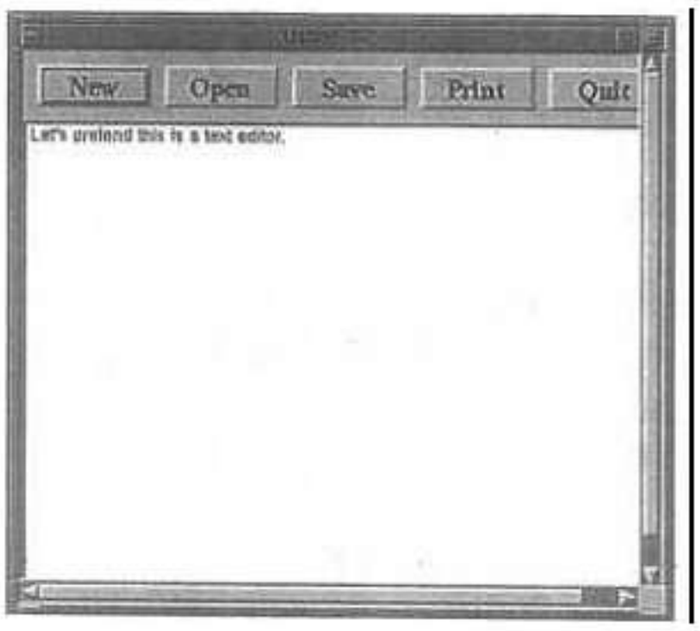


图 5-5　一个使用 QScrollView 的简单文本编辑器

注意，开始时看不到任何滚动条。但是，如果你稍微缩小一点窗口，就会出现滚动条。

这个例子中所使用的 QMultiLineEdit 类有内置滚动条功能。这能够减少你的工作量，因为你不必再向 QMultiLineEdit 对象手工添加滚动条。当文本行太多或是行太长而不能显示在文本区域内时，滚动条就会显示出来。为了能够看到这一功能，点击一下文本区域，然后按回车键直到出现滚动条为止。然后再水平添加文本直到水平滚动条出现为止。这时窗口看起来与图 5-6 类似。

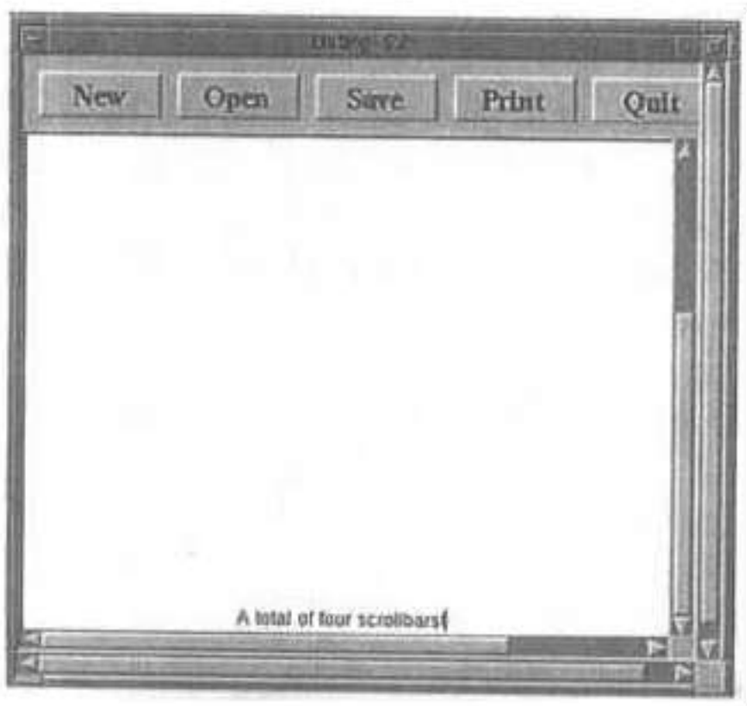


图 5-6　QMultiLineEdit 具有滚动条功能，实际上这个程序已经有 4 个滚动条

你已经看到：在创建大型应用程序时滚动条很有帮助。更重要的是，在 Qt 应用程序中实现滚动条功能并不困难。

5.2 添加菜单

在应用程序的顶部有几个下拉菜单将使界面更加友好（尽管这不是必须的）。在这一学时后面部分你将看到 QMainWindow 类，它能够自动创建标准菜单。但是，首先你应该学习怎样创建我们自己的用户菜单。

为了创建下拉菜单，需要使用两个 Qt 类：QMenuBar 和 QPopupMenu。QPopupMenu 代表单个菜单，而 QMenuBar 则表示整个菜单条。为了创建一个具有几个菜单项的菜单条，首先你需要创建几个 QPopupMenu 对象，并向它们添加条目。然后，再创建一个 QMenuBar 对象，并将 QPopupMenu 对象添加到其中。这样的例子如程序清单 5-3 所示。

程序清单 5-3 一个简单的菜单例子

```
1: #include <qapplication.h>
2: #include <qpushbutton.h>
3: #include <qfont.h>
4: //We include the header-files we need to
5: //use the QMenuBar and QPopupMenu classes:
6: #include <qpopupmenu.h>
7: #include <qmenubar.h>
8:
9: class MyMainWindow : public QWidget
10: {
11: public:
12:     MyMainWindow();
13: private:
14:     //Locate memory for one QPopupMenu and one
15:     //QMenuBar object:
16:     QPopupMenu *file;
17:     QMenuBar *menubar;
18: };
19:
20: MyMainWindow::MyMainWindow()
21: {
22:     setGeometry( 100, 100, 300, 300 );
23:
24:     //Create the popupmenu and add an item to it.
25:     //We also connect the item to the quit() slot
26:     //of qApp. Note that we don't define any parent
27:     //here. That's because we will insert it in the
28:     //QMenuBar object later:
29:     file = new QPopupMenu();
30:     file->insertItem( "Quit", qApp, SLOT( quit() ) );
31:
32:     //Create the menubar and insert the QPopupMenu
33:     //object in it:
34:     menubar = new QMenuBar( this );
35:     menubar->insertItem( "File", file );
36: }
```

```
37:
38: void main( int argc, char **argv )
39: {
40:     QApplication a(argc, argv);
41:     MyMainWindow w;
42:     a.setMainWidget( &w );
43:     w.show();
44:     a.exec();
45: }
```

菜单创建过程非常简单。首先你创建 QPopupMenu 对象，并向其中添加条目（这个例子中为 Quit）。这个条目被连接到 qAPP 的 quit()槽。然后，创建一个 QMenuBar 对象，并将 QPopupMenu 插入到其中。这段代码所创建的菜单条具有单个菜单，标签为 File。File 菜单具有一个条目，叫作 Quit。该程序显示结果如图 5-7 所示。

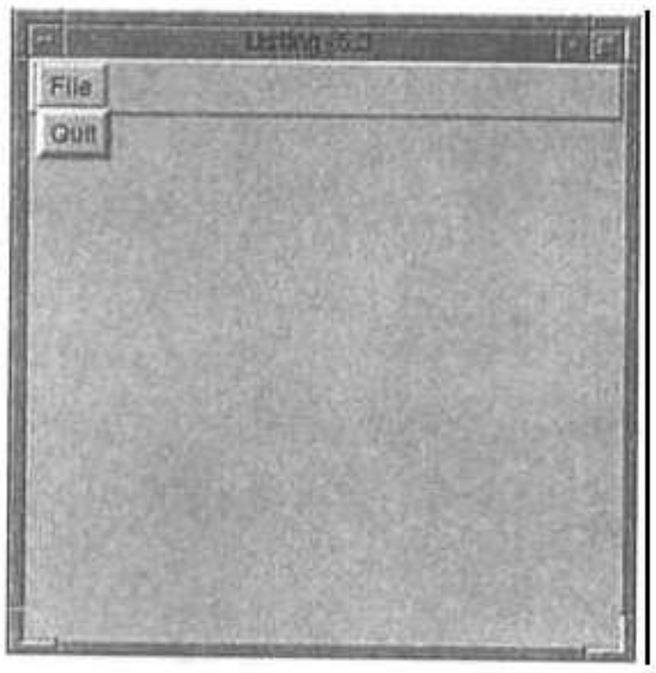


图 5-7　具有一个菜单条的窗口，该菜单条拥有单个条目

当然，按照同样的过程，你可以添加多个条目和菜单。在程序清单 5-4 中，菜单条中被添加了多个菜单和条目。

程序清单 5-4　　具有多个条目的菜单条

```
 1: #include <qapplication.h>
 2: #include <qpushbutton.h>
 3: #include <qfont.h>
 4: #include <qpopupmenu.h>
 5: #include <qmenubar.h>
 6:
 7: class MyMainWindow : public QWidget
 8: {
 9: public:
10:     MyMainWindow();
11: private:
12:     QPopupMenu *file;
13:     QPopupMenu *menu2;
14:     QPopupMenu *menu3;
15:     QMenuBar *menubar;
16: };
17:
18: MyMainWindow::MyMainWindow()
19: {
20:     setGeometry( 100, 100, 300, 300 );
```

```
21:
22:     file = new QPopupMenu();
23:     file->insertItem( "Quit", qApp, SLOT( quit() ) );
24:     file->insertItem( "Open" );
25:     file->insertItem( "Save" );
26:     file->insertItem( "New" );
27:
28:     menu2 = new QPopupMenu();
29:     menu2->insertItem( "Item-2.1" );
30:     menu2->insertItem( "Item-2.2" );
31:     menu2->insertItem( "Item-2.3" );
32:     menu2->insertItem( "Item-2.4" );
33:
34:     menu3 = new QPopupMenu();
35:     menu3->insertItem( "Item-3.1" );
36:     menu3->insertItem( "Item-3.2" );
37:     menu3->insertItem( "Item-3.3" );
38:     menu3->insertItem( "Item-3.4" );
39:
40:     menubar = new QMenuBar( this );
41:     menubar->insertItem( "File", file );
42:     menubar->insertItem( "Menu2", menu2 );
43:     menubar->insertItem( "Menu3", menu3 );
44: }
45:
46: void main( int argc, char **argv )
47: {
48:     QApplication a(argc, argv);
49:     MyMainWindow w;
50:     a.setMainWidget( &w );
51:     w.show();
52:     a.exec();
53: }
```

第 22 行到 36 行，创建 File 菜单，并向其中添加条目，但只有一个条目，Quit，被连接到一个槽（qApp->quit()）。第 28 行到第 38 行创建另外两个菜单，并向它们添加条目。所有这些条目都未被连接到任何槽。然后，在第 40 行创建实际的菜单条，并在第 41、42 和 43 行分别把这三个菜单插入到菜单条中。该程序显示如图 5-8 所示。



图 5-8 这里向菜单条中插入更多条目，但由于它们未被连接到任何槽，所以它们没有任何作用

现在，你已经知道怎样向应用程序添加菜单条。菜单条能够为应用程序添加许多功能，

并且创建它们并不困难，这正如你已经看到的。

5.3　使用 QMainWindow 部件

GUI 应用程序的开发通过这些年的发展，已经形成一套图形应用程序界面标准。这一标准多数应用于 Microsoft 平台，但是，后来它也越来越多地应用于 UNIX 平台。你很可能想遵守这一标准创建应用程序，以使用户能够更容易地使用你的应用程序。QMainWindow 类是你实现这一愿望的一个很好助手。它能使你很容易地添加工具条和菜单，并使这些工具条和菜单的外观看起来更标准。如果你想创建标准应用程序，QMainWindow 类能够减轻你大量的工作量。

5.3.1　添加菜单、按钮和中心部件

首先来看一个基于 QMainWindow 类的简单程序（如程序清单 5-5 所示）。这个程序是一个虚拟文本编辑器，它具有菜单、工具条和一个大的文本输入区域。这个例子向你演示创建专业程序外观是多么容易。

程序清单 5-5　　一个基于 QMainWindow 的程序

```
1: #include <qapplication.h>
2: #include <qmainwindow.h>
3: #include <qpopupmenu.h>
4: #include <qmenubar.h>
5: #include <qtoolbar.h>
6: #include <qtoolbutton.h>
7: #include <qpixmap.h>
8: #include <qmultilineedit.h>
9:
10: //We include a few pixmaps, which
11: //we will use for the buttons on
12: //the toolbar.
13: #include "fileopen.xpm"
14: #include "filesave.xpm"
15: #include "fileprint.xpm"
16:
17: class MyMainWindow : public QMainWindow
18: {
19: public:
20:     MyMainWindow();
21: private:
22:         //We create four menus:
23:     QPopupMenu *file;
24:     QPopupMenu *edit;
25:     QPopupMenu *options;
26:     QPopupMenu *help;
27:         //We create one toolbar
28:         //and three buttons, which
29:         //we will add to the toolbar.
30:     QToolBar *toolbar;
31:     QToolButton *b1, *b2, *b3;
32:         //We create three QPixmap objects,
```

```
33:         //one for each .xpm file.
34:     QPixmap openicon, saveicon, printicon;
35:         //At last, we create a QMultiLineEdit
36:         //object. We make this widget the "central
37:         //widget" in the application.
38:     QMultiLineEdit *centralwidget;
39: };
40:
41: MyMainWindow::MyMainWindow()
42: {
43:     setGeometry( 100, 100, 400, 400 );
44:
45:     //Start menu definition:
46:     file = new QPopupMenu( this );
47:     file->insertItem( "New" );
48:     file->insertItem( "Open" );
49:     file->insertItem( "Save" );
50:     file->insertSeparator();
51:     file->insertItem( "Exit", qApp, SLOT( quit() ) );
52:
53:     edit = new QPopupMenu( this );
54:     edit->insertItem( "Undo" );
55:     edit->insertItem( "Redo" );
56:     edit->insertSeparator();
57:     edit->insertItem( "Cut" );
58:     edit->insertItem( "Copy" );
59:     edit->insertItem( "Paste" );
60:     edit->insertSeparator();
61:     edit->insertItem( "Select All" );
62:
63:     options = new QPopupMenu( this );
64:     options->insertItem( "Preferences" );
65:
66:     help = new QPopupMenu( this );
67:     help->insertItem( "Contents" );
68:     help->insertSeparator();
69:     help->insertItem( "About" );
70:     //End menu definition.
71:
72:     //Start insertion of menus in
73:     //QMainWindow:
74:     menuBar()->insertItem( "File", file );
75:     menuBar()->insertItem( "Edit", edit );
76:     menuBar()->insertItem( "Options", options );
77:     menuBar()->insertItem( "Help", help );
78:
79:     //Define three pixmaps:
80:     openicon = QPixmap( fileopen );
81:     saveicon = QPixmap( filesave );
82:     printicon = QPixmap( fileprint );
83:
84:     //Create a toolbar and add three
85:     //buttons to it. We connect the
86:         //buttons to three slots that not
87:         //exist (but feel free to implement
88:     //them!):
89:     toolbar = new QToolBar( this );
90:     b1 = new QToolButton( openicon, "Open File", "Open File",
```

```
 91:                     this, SLOT( open() ), toolbar );
 92:     b2 = new QToolButton( saveicon, "Save File", "Save File",
 93:                     this, SLOT( save() ), toolbar );
 94:     b3 = new QToolButton( printicon, "Print File", "Print File",
 95:                     this, SLOT( print() ), toolbar );
 96:
 97:     //Set centralwidget to the
 98:     //applications central widget.
 99:     centralwidget = new QMultiLineEdit( this );
100:     setCentralWidget( centralwidget );
101: }
102:
103: void main( int argc, char **argv )
104: {
105:     QApplication a(argc, argv);
106:     MyMainWindow w;
107:     a.setMainWidget( &w );
108:     w.show();
109:     a.exec();
110: }
```

如上所述，这个程序基于 QMainWindow 类。这使得创建菜单和按钮有一点不同。例如，你不需创建任何 QMenuBar 对象，而是由程序第 74 行到第 77 行中的 QMainWindow 对象实现。程序中的第 80 行到第 82 行定义 3 个 QPixmap 对象，这些位图用在工具条中。



在第 9 学时“创建简单图形”中，你将学到另外一种使用位图的方法，但是，将位图用做资源（像这个例子中这样）会更好。

第 89 行创建工具条，第 90 行到第 95 行为该工具条创建 3 个按钮。表 5-1 说明传递给 QToolButton 构造函数的参数。

表 5-1　　QToolButton 构造函数参数

参　数	描　述
openicon	QPixmap 对象，它所具有的位图用于装饰按钮
"Open File"	文本字符串，当把鼠标指针放置在按钮上几秒钟后将显示该文本
"Open File"	文本字符串，当把鼠标指针放置在按钮时，将在状态条内显示该文本（如果状态条存在的话）
this	一个对象，按钮需要连接该对象所具有的槽
SLOT(open())	实际槽
toolbar	父部件



第 90 行到第 95 行所使用的槽是不存在的，这就是程序运行时你将得到错误信息的原因。这些错误信息告诉你被定义的槽不存在。尽管当未定义槽被调用时不会发生任何事情，但程序仍能够继续运行

第 99 行创建 QMultiLineEdit 对象。第 100 行将该对象设置到中心部件。在基于 QMainWindow 的程序中，中心部件是一个窗口，所添加的菜单、工具条和状态条环绕在其

周围。定义为中心部件的部件是程序的最主要部分（如文本编辑器的文本区域）。如果创建一个绘图程序，中心部件很可能是用户的绘图区域。如果在一个 Tetris 游戏中，中心部件则是正在玩的部件（这种情况下，你已经自己创建了一个部件）。这个例子中使用 QMultiLineEdit 部件作为中心部件，因此，程序看起来像一个文本编辑器。该程序显示结果如图 5-9 所示。



图 5-9 不用很多行代码就能够创建一个与 Microsoft Office 软件包中的程序相似的外观

5.3.2 添加状态条

状态条是在很多应用程序窗口的下端能够看到的一个小条。它根据用户当前的不同操作向用户显示不同的信息。为程序清单 5-5 中的程序添加一个状态条是一件非常简单的工作。只要把下面一行代码添加到类构造函数中即可：

```
statusBar( );
```

这是 QMainWindow 的一个成员函数。它在窗口下端创建一个简单的状态条。现在，这个程序外观如图 5-10 所示。



图 5-10 当前所显示的为 Open 按钮的状态信息

如果将鼠标指针保持在一个按钮上不动，一条信息就将显示在状态条中。该信息由传

递给 QToolButton 构造函数的第 3 个参数定义（这里为 Open File）。但是，通常你会给出一个更详细的描述，如点击这里打开文件。

状态条用于简短地告诉用户当他执行某种用户事件后将会发生什么事情。在程序的状态条中，为每个条目显示一个简短的解释文本，能够使程序更加容易使用和理解。

除解释外，状态条还有其他用途。例如，可以使用状态条显示文本编辑器中的当前行号，或者当前所打开的文件名称等。一个状态条能够改善程序的性能。使用它所能做的事情完全取决于你的想象能力。

5.4　小　结

这一学时使你在 Qt 学习的道路上又前进了一步。我们讨论了 Qt 的一些基本内容——这些内容在你创建易于使用的 GUI 应用程序时非常有价值。

你学习了怎样使用滚动条、菜单、工具条和状态条。这些几乎是每一个人都希望在程序中使用的项目。所有这些项目都能够改善程序性能。滚动条使程序能够更加容易地应用在不同的桌面上，使用户在根据屏幕大小调整窗口尺寸时不至于错过程序中的任何一部分。使用菜单和工具条能够很容易地组织程序功能和选项，并使程序变得更加容易理解（使用）。

你已经看到了 QMainWindow 类，QMainWindow 可能是 Qt 中功能最强大的一个类。事实上，大多情况下（用户界面不是非常特殊时）都要使用 QMainWindow 类。使用 QMainWindow 类能够使标准 GUI 设置变得更加快捷和简单。

前面已经讲过，这一学时覆盖 Qt 编程的很多重要方面。因此，你要确实理解其中所有内容，并完成下面的“问题与答案”和“作业”两部分内容。

5.5　问题与答案

问：我正在创建一个滚动窗口，我使用 addChild()函数添加了很多对象。但是，QScrollView 类似乎只注意到我添加的第一个对象。如果不影响第一个对象的话，滚动条就不出现。这是为什么？

答：解决这一问题最简单的方法就是将你的子部件添加到一个 QWidget 对象中，然后再使用 addChild()函数将 QWidget 对象作为一个子部件添加到 QScrollView 中。详细细节请参看程序清单 5-2。

问：我创建一个菜单条，但是我所添加的菜单并没有出现在窗口上。这是为什么？

答：要确保是使用 QMenuBar::insetItem()函数将所有菜单添加到 QMenuBar 对象中。

问：我基于 QMainWindow 创建一个类，但是我的菜单没有显示在窗口上。这是为什么？

答：你必须使用 menuBar()->insetItem()函数添加所有的菜单。

问：当我编译程序清单 5-5 时，编译器报告未找到位图文件。这是为什么？

答：为了解决这一问题，你必须将位图文件 fileopen.xpm、filesave.xpm 和 fileprint.xpm 拷贝到编译程序所在目录。这些文件可以从 Qt 安装目录中的 examples/application 目录下找到。

5.6 作 业

希望你完成下面的问题和练习，这有助于你牢记本学时所学内容。完成这些问题和练习，能够帮助你理解这一学时所讲述的 Qt 基本类的工作方式。

5.6.1 测验

1. QscrollView 有什么用途？
2. 哪个成员函数用于向 QScrollView 类添加对象？
3. 按需滚动是什么意思？
4. 什么是 QMenuBar 和 QPopupMenu？
5. 什么时候需要调用 QMenuBar::insertItem ()？
6. 什么是 QToolBar 和 QToolButton？
7. QMainWindow 类适用于做什么？
8. 什么是 QMainWindow 对象中的中心部件？
9. 向基于 QMainWindow 的类添加工具条时是否需要调用特殊的函数？
10. 什么是 QPixmap？

5.6.2 练习

1. 基于 QScrollView 类编写一个程序，创建一个 QWidget 对象，并向它添加几个部件（可以查阅 Qt Reference Document 以找到你还未学习过的部件）。然后，将 QWidget 对象添加到 QScrollView。
2. 向练习 1 的执行结果中添加几个菜单，确保使菜单成为 QWidget 对象的子部件。
3. 启动一个流行的文本编辑器，如 kedit 或 Notepad，然后基于 QMainWindow 编写一个程序，它模拟文本编辑器的外观。并使你的菜单、工具条、状态条看起来尽可能像真正的文本编辑器一样（你可能需要查阅 Qt Reference Document）。

第二部分

重要的 Qt 部件

第 6 学时　认识 Qt 部件的第 1 课

第 7 学时　认识 Qt 部件的第 2 课

第 8 学时　认识 Qt 部件的第 3 课

第 9 学时　创建简单图形

第 10 学时　理解 Qt 对话框

第 6 学时　认识 Qt 部件的第 1 课

这一学时，你将深入了解 Qt 部件，尤其是按钮、标签和表的用法。

在 GUI 应用程序中，按钮和标签的使用都非常频繁。因此，你必须了解怎样使用它们来创建即使是最简单的应用程序。在所有现代 GUI 库中，按钮和标签都占有重要的位置——Qt 中也是这样。

相反，表在 GUI 应用程序中并不常用。但是，它们具有很重要的功能（如果正确使用的话）。因此，在应用程序中，在所有你感到需要的地方都应该使用表。在这一学时的最后你将学到更多关于表方面的知识。

6.1　使用按钮

按钮可能是最常用的 GUI 对象。Qt 提供 3 种类型的按钮：

- 按钮（Push Button）；
- 单选按钮（Radio Button）；
- 复选按钮（Check Button）。

第 1 种按钮用于产生某种事件，而单选按钮和复选按钮则用于做一些选择。

6.1.1　按钮

QPushButton 类可能是你创建 Qt 应用程序过程中使用最多的 Qt 类。在本书前面几个学时中已经创建了一些按钮。但是，现在我们来进一步研究 QPushButton 的工作方式，以及哪些成员函数对你来说是有用的。

如前所述，按钮由 QPushButton 类创建。程序清单 6-1 是一个简单例子：

程序清单 6-1　　QPushButton 例子

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qpushbutton.h>
4:
5: class MyMainWindow : public QWidget
6: {
7: public:
8:         MyMainWindow();
```

```
 9: private:
10:         QPushButton *b1;
11: };
12:
13: MyMainWindow::MyMainWindow()
14: {
15:         setGeometry( 100, 100, 200, 100 );
16:         b1 = new QPushButton( "This is a push button!", this );
17:         b1->setGeometry( 10, 10, 180, 80 );
18:         b1->setFont( QFont( "Times", 16, QFont::Bold ) );
19: }
20:
21: void main( int argc, char **argv )
22: {
23:         QApplication a(argc, argv);
24:         MyMainWindow w;
25:         a.setMainWidget( &w );
26:         w.show();
27:         a.exec();
28: }
```

第 10 行用于为 QPushButton 对象分配内存。第 16 行创建按钮（也就是执行 QPushButton 的一个构造函数）。这里并设置按钮标签（第一个参数）及其父对象（第二个参数）。第 17 行设置按钮的放置位置和大小。在这里，将按钮的左上角放置在相对于其父对象（实际窗口）左上角下面 10 个像素和左边 10 个像素的位置。第 18 行，设置按钮上标签的大小和样式，这一点前面已经讨论过。程序清单 6-1 所创建的按钮显示结果如图 6-1 所示。

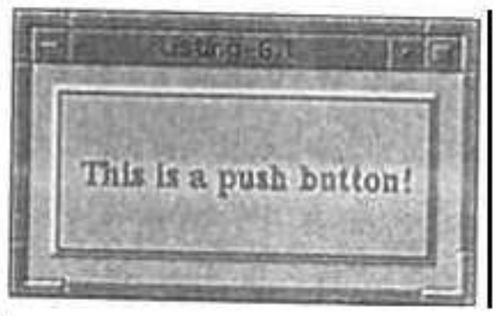


图 6-1 程序清单 6-1 所创建的按钮

当需要将按钮连接于某个事件时，将 QPushButton::clicked()信号连接到一个槽即可。你也可以象下面这样使用 QPushButton::setPixmap()函数为按钮添加一个位图标签：

```
QPixmap pixmap("somepixmap.xpm" ) ;
b1->setPixmap(pixmap ) ;
```

这里，somepixmap.xpm 是你所选择的位图。图 6-2 显示一个具有位图标签的按钮。

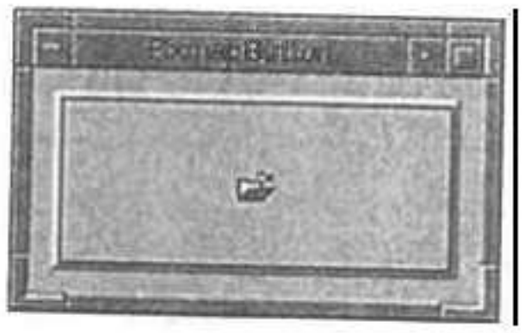


图 6-2 具有位图标签的按钮

当使用按钮时，很可能还需要使用 QPushButton::setDefault()函数。如果在一个窗口中有多个按钮时，这个函数就很有用。应用 QPushButton::setDefault()的按钮将变为默认按钮——即用户按回车键时将点击的按钮。

6.1.2　单选按钮

当需要用户从几个选项中选择一项（并且只有一项）时，就要用到单选按钮。你可以像程序清单 6-2 中那样使用 QButtonGroup 和 QRadioButton 类创建一组单选按钮：

程序清单 6-2　　QRadioButton 例子

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qbuttongroup.h>
4: #include <qradiobutton.h>
5:
6: class MyMainWindow : public QWidget
7: {
8: public:
9:     MyMainWindow();
10: private:
11:     QButtonGroup *group;
12:     QRadioButton *b1;
13:     QRadioButton *b2;
14:     QRadioButton *b3;
15: };
16:
17: MyMainWindow::MyMainWindow()
18: {
19:     setGeometry( 100, 100, 150, 140 );
20:
21:     group = new QButtonGroup( "Options", this );
22:     group->setGeometry( 10, 10, 130, 120 );
23:
24:     b1 = new QRadioButton( "Choice 1", group );
25:     b1->move( 20, 20 );
26:     b2 = new QRadioButton( "Choice 2", group );
27:     b2->move( 20, 50 );
28:     b3 = new QRadioButton( "Choice 3", group );
29:     b3->move( 20, 80 );
30:
31:     group->insert( b1 );
32:     group->insert( b2 );
33:     group->insert( b3 );
34: }
35:
36: void main( int argc, char **argv )
37: {
38:     QApplication a(argc, argv);
39:     MyMainWindow w;
40:     a.setMainWidget( &w );
41:     w.show();
42:     a.exec();
43: }
```

第 11 行到第 14 行为一个 QButtonGroup 对象和三个 QRadioButton 对象分配内存。然后在第 21、22 行设置 QButtonGroup 对象。传递给 QButtonGroup 构造函数的第一个参数（第 21 行）表示按钮组标题。第 22 行使用 setGeometry()函数设置按钮组的几何位置。之后，

创建 3 个单选按钮，并在第 24 到 29 行中对它们进行设置。注意，单选按钮组用做这些按钮的父对象。另外也要注意，这里使用 QRadioButton::move()函数设置这些按钮的位置。尽管也可以使用 setGeometry()函数，但不推荐使用这种方法，因为你不知道这些按钮的宽度和高度。最后，使用 QButtonGroup::insert()函数将这些按钮插入到按钮组中（第 31、32 和 33 行）。图 6-3 显示出程序清单 6-2 的执行结果。

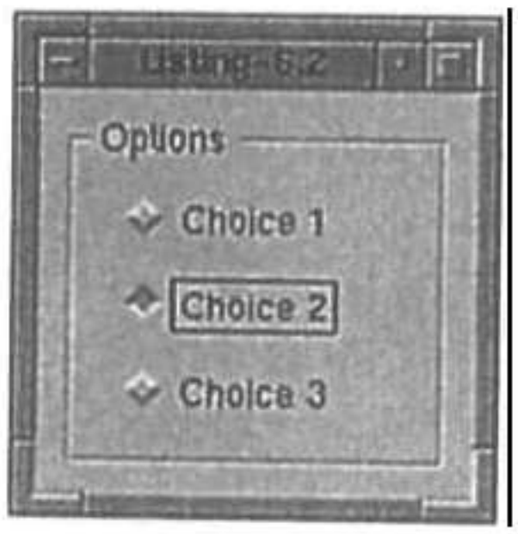


图 6-3　具有 3 个 QRadioButton 对象的 QButtonGroup 对象

注意，你每次只能按下（选取）1 个单选按钮。这是单选按钮与复选按钮之间的区别。

当选取（点击）一个按钮时，将发射 QRadioButton::isChecked()信号。但是，与按钮（Push Button）不同的是，单选按钮通常不用来使程序做某些可视化的事情，而是用于做一些选择。例如，你可以使用几个单选按钮使用户选择拼写检查器中的语言种类。之后，连接到 QRadioButton::isChecked()信号的槽并不引起程序视觉上的变化，而是调用一个函数，它将语言设置为用户最近所选择的语言种类。

6.1.3　复选按钮

当你希望用户能够从多个选项中选择超过一种以上的选择时，就应该使用复选按钮。使用 QButtonGroup 和 QCheckBox 类可以创建一组复选按钮。程序清单 6-3 即是一个例子：

程序清单 6-3　　一个 QCheckBox 例子

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qbuttongroup.h>
4: #include <qcheckbox.h>
5:
6: class MyMainWindow : public QWidget
7: {
8: public:
9:     MyMainWindow();
10: private:
11:     QButtonGroup *group;
12:     QCheckBox *b1;
13:     QCheckBox *b2;
14:     QCheckBox *b3;
15: };
16:
17: MyMainWindow::MyMainWindow()
18: {
```

```
19:     setGeometry( 100, 100, 150, 140 );
20:
21:     group = new QButtonGroup( "Options", this );
22:     group->setGeometry( 10, 10, 130, 120 );
23:
24:     b1 = new QCheckBox( "Choice 1", group );
25:     b1->move( 20, 20 );
26:     b2 = new QCheckBox( "Choice 2", group );
27:     b2->move( 20, 50 );
28:     b3 = new QCheckBox( "Choice 3", group );
29:     b3->move( 20, 80 );
30:
31:     group->insert( b1 );
32:     group->insert( b2 );
33:     group->insert( b3 );
34: }
35:
36: void main( int argc, char **argv )
37: {
38:     QApplication a(argc, argv);
39:     MyMainWindow w;
40:     a.setMainWidget( &w );
41:     w.show();
42:     a.exec();
43: }
```

这里，第 11 行到第 14 行为按钮组和复选按钮分配内存。第 21 行和 22 行设置按钮组，第 24 到 29 行设置复选按钮。与单选按钮例子一样，使用 QButtonGroup::insert()函数将复选按钮插入到按钮组中（第 31、32 和 33 行）。图 6-4 为程序清单 6-3 的执行结果。

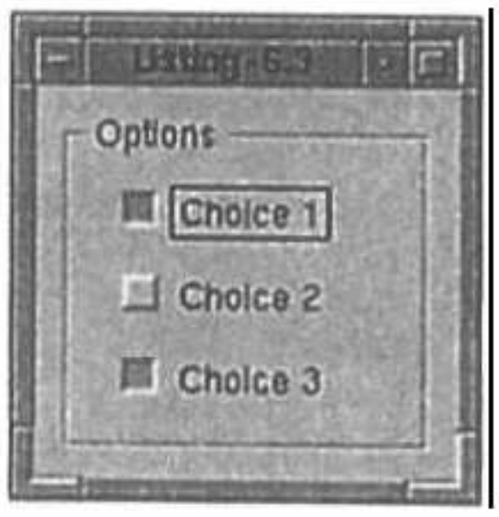


图 6-4　具有 3 个 QCheckBox 对象的 QButtonGroup 对象例子

与单选按钮例子一样，你通过 isChecked()信号（QCheckBox::isChecked()）检查 QCheckBox 对象是否被选取。如前所述，复选按钮使用户能够做出多个选择。因此，在前一节为单选按钮所举的拼写检查器例子中你不能使用复选按钮。如果这样做，用户将能够同时选择 English、French 和 German，这将导致拼写检查器出现混乱。

一个说明复选按钮比较有用的实际例子是让用户在一个文本编辑器中选择文本的样式。在这种情况下，完全有可能同时选择粗体、斜体和下划线属性。但是，如果这时使用单选按钮，用户将只能一次选择一种格式化方法，这显然不是我们所希望的。

使用按钮非常直观，你现在已经学习了 Qt 所提供的 3 种按钮类型。如果需要更多信息，可查阅 Qt Reference Document。

6.2　创建标签

你经常会需要以标签的形式向应用程序添加文本。在程序中，标签常用来显示简短信息或者是部件的说明。例如，你可以使用标签定义应该在文本框中所输入的内容。在 Qt 中，使用 QLabel 类，偶尔也使用 QLCDNumber 类，创建标签。

6.2.1　QLabel

QLabel 类用于显示简单的文本。它是一个最基本也是一个很有用的部件。程序清单 6-4 给出一个简单的例子：

程序清单 6-4　　QLabel 例子

```
 1: #include <qapplication.h>
 2: #include <qlabel.h>
 3:
 4: class MyMainWindow : public QWidget
 5: {
 6: public:
 7:     MyMainWindow();
 8: private:
 9:     QLabel *text;
10: };
11:
12: MyMainWindow::MyMainWindow()
13: {
14:     setGeometry( 100, 100, 170, 100 );
15:
16:     text = new QLabel( this );
17:     text->setGeometry( 10, 10, 150, 80 );
18:     text->setText( "This is a \nQLabel object." );
19:     text->setAlignment( AlignHCenter | AlignVCenter );
20: }
21:
22: void main( int argc, char **argv )
23: {
24:     QApplication a(argc, argv);
25:     MyMainWindow w;
26:     a.setMainWidget( &w );
27:     w.show();
28:     a.exec();
29: }
```

图 6-5 为程序清单 6-4 的执行结果。



图 6-5　具有简单的 QLabel 对象的窗口

注意第 19 行中怎样设置文本的对齐方式。这是 Qt 中设置对齐方式的标准方法。AlignHCenter 和 AlignVCenter 是 Qt 定义。表 6-1 列出了所有对齐设置的定义。

表 6-1　　对齐设置定义

定　义	描　述
AlignTop	将文本添加到 QLabel 对象的上部
AlignBottom	将文本添加到 QLabel 对象的下部
AlignLeft	沿着 QLabel 对象的左边添加文本
AlignRight	沿着 QLabel 对象的右边添加文本
AlignHCenter	将文本添加到 QLabel 对象的水平中心位置
AlignVCenter	将文本添加到 QLabel 对象的垂直中心位置
AlignCenter	这与 AlignVCenter 和 AlignHCenter 的设置结果相同
WordBreak	自动断字
ExpandTabs	扩展制表符

注意，传递给 QLabel::setAlignment()的参数被“或”(|) 运算符所分隔。当你想添加多个参数时需要使用这种方法，也只能使用 | 运算符，分隔它们。



你也可以使用 QLabel 显示位图和动画。这一功能由 QLabel::setPixmap() 和 QLabel::setMovie()函数实现。

6.2.1　QLCDNumber

如果程序中需要显示数字信息，你选择使用 QLCDNumber 类是非常恰当的。与 QLabel 一样，QLCDNumber 类常用来向用户显示日期一类的短信息（这里指数字信息）。程序清单 6-5 给出了一个使用 QLCDNumber 对象的例子：

程序清单 6-5　　一个 QLCDNumber 例子

```
1: #include <qapplication.h>
2: #include <qlcdnumber.h>
3:
4: class MyMainWindow : public QWidget
5: {
6: public:
7:         MyMainWindow();
8: private:
9:         QLCDNumber *number;
10: };
11:
12: MyMainWindow::MyMainWindow()
13: {
14:         setGeometry( 100, 100, 170, 100 );
```

```
15:         number = new QLCDNumber( this );
16:         number->setGeometry( 10, 10, 150, 80 );
17:         number->display( 12345 );
18: }
19:
20: void main( int argc, char **argv )
21: {
22:         QApplication a(argc, argv);
23:         MyMainWindow w;
24:         a.setMainWidget( &w );
25:         w.show();
26:         a.exec();
27: }
```

首先，在第 9 行中为 QLCDNumber 对象分配内存。然后在第 15 行中创建该对象，第 16 行设置其几何位置。

之后，使用 QLCDNumber::display()函数设置所显示的数字（第 17 行）。

图 6-6 显示出程序清单 6-5 的运行结果。

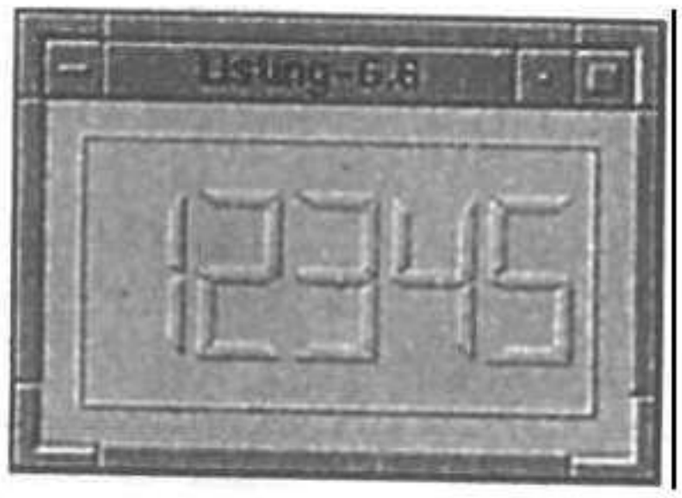


图 6-6 一个具有 QLCDNumber 对象的窗口

当使用 QLCDNumber 类时，其他可能有用的函数如表 6-2 所示。

表 6-2 QLCDNumber 成员函数

函 数	描 述
setNumDigits()	设置所显示的位数
setBinMode()	要求以二进制方式显示
setOctMode()	要求以八进制方式显示
setHexMode()	要求以十六进制方式显示
setDecMode()	要求以十进制方式显示（默认）
setSmallDecimalPoint()	其参数为 TRUE 或 FALSE，它决定小数点是单独占一位空间还是在两个位之间。换句话说，如果函数参数为 TRUE，小数点将占用比平常更少的空间
setSegmentStyle()	改变所显示数字的外观，可以将 Outline、Filled 或 Flat 参数传递给它
checkOverFlow()	调用这个函数检查给定值能否在显示区域内显示，（QLCDNumber 也发射 overflow()信号，你可以将它连接到一个处理这一情况的槽）

记住，QLCDNumber 常与 QSlider 对象一起使用。这样的例子请参看第 4 学时“信号与槽”中的程序清单 4-1。

6.3　表

Qt 提供的 QTableView 类用于创建表。它是一个很有用的类，但是，其定义也非常抽象。因此，为了很好地使用它，你需要从 QTableView 类派生一个新类。

6.3.1　创建简单的网格

当基于 QTableView 创建类时，你需要实现 paintCell()函数。这个函数在 QTableView 类中没有定义，但该类使用它绘制表元。因此，你需要实现该函数，并由它处理表元绘制操作。这你听起来可能感到有点难，其实完全不是这样。程序清单 6-6 给出一个简单的例子，说明怎样使用 QTableView 创建网格。



当你实现 paintCell()函数功能时，你必然要用到 QPainter 类的一些绘图功能。第 9 学时“创建简单图形”和第 15 学时“深入学习图形”中将详细介绍该类及其方法。

程序清单 6-6　　使用 QTableView 类创建简单的网格

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qtableview.h>
 4: #include <qpainter.h>
 5:
 6: class MyMainWindow : public QTableView
 7: {
 8: public:
 9:     MyMainWindow();
10: private:
11:     void paintCell( QPainter *, int, int );
12: };
13:
14: MyMainWindow::MyMainWindow()
15: {
16:     setGeometry( 100, 100, 300, 300 );
17:
18:     //Set the number of columns:
19:     setNumCols( 6 );
20:     //Set the number of rows:
21:     setNumRows( 10 );
22:     //Set the width of the cells:
23:     setCellWidth( 50 );
24:     //Set the height of the cells:
25:     setCellHeight( 30 );
26: }
27:
```

```
28: //Here is the simple implementation of paintCell():
29: void MyMainWindow::paintCell( QPainter* p, int row, int col )
30: {
31:     //Find out height and width of the cells:
32:     int x = cellWidth( col ) ;
33:     int y = cellHeight( row );
34:
35:     //Two lines are enough for creating a cell:
36:     p->drawLine( x, 0, x, y );
37:     p->drawLine( 0, y, x, y );
38: }
39:
40: void main( int argc, char **argv )
41: {
42:     QApplication a(argc, argv);
43:     MyMainWindow w;
44:     a.setMainWidget( &w );
45:     w.show();
46:     a.exec();
47: }
```

这个例子绘制一个简单的具有 6 列（第 19 行）、10 行（第 21 行）的网格。所有表元都具有相同的尺寸：50 像素宽（第 23 行）、30 像素高（第 25 行）。

从第 29 行开始定义 paintCell()函数。正如你所看到的，这是一个非常简单的函数，总共只有 4 行代码组成。第 32 行和第 33 行调用 cellWidth()和 cellHeight()函数分别确定当前表元（即将要绘制的表元）的宽度和高度。第 36 行和第 37 行实际绘制表元。注意，只绘制低端的水平线和右边的垂直线。



当看到第 36 行和 37 行时，你可能认为这两行总是在相同位置绘图。但是，事实并不是这样——这由 QTableView 决定。QTableView 每次都设置 QPainter 的绘图区域，因此(0,0)是每个表元的角。

图 6-7 为程序清单 6-6 运行结果。

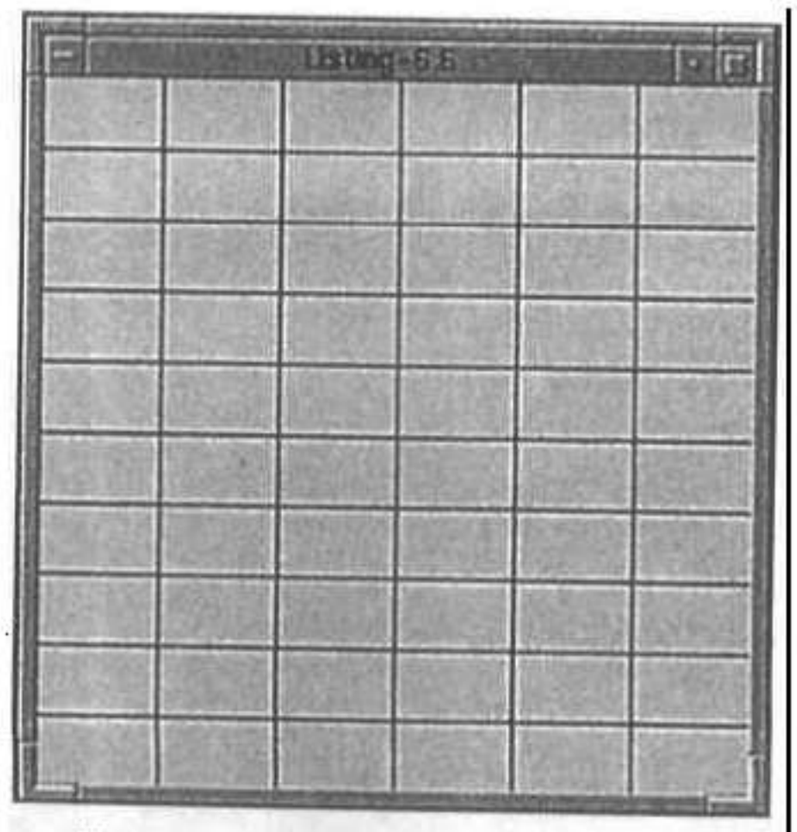


图 6-7　一个基于 QTableView 的简单程序

6.3.2　添加文本和点击选择功能

尽管程序清单 6-6 绘制了一个很好的网格，但它还没有什么用处。你很可能想向表元中添加一些文本信息。程序清单 6-7 包含了文本和点击选择功能。

程序清单 6-7　　具有文本和点击选择功能的表格

```
#include <qapplication.h>
#include <qwidget.h>
#include <qtableview.h>
#include <qpainter.h>

class MyMainWindow : public QTableView
{
public:
    MyMainWindow();
private:
    void paintCell( QPainter *, int, int );
    void mousePressEvent( QMouseEvent * );
    int curRow, curCol;
};

MyMainWindow::MyMainWindow()
{
    setGeometry( 100, 100, 300, 300 );

    setNumCols( 12 );
    setNumRows( 20 );
    setCellWidth( 80 );
    setCellHeight( 30 );

    //The following function call makes
    //both horizontal and vertical scrollbars
    //appear:
    setTableFlags(    Tbl_vScrollBar |
                      Tbl_hScrollBar );
    //This line sets changes the colors
    //of the cells:
    setBackgroundMode( PaletteBase );

    curRow = curCol = 0;
}

void MyMainWindow::paintCell( QPainter* p, int row, int col )
{
    int x = (cellWidth( col ) - 1);
    int y = (cellHeight( row ) - 1);

    p->drawLine( x, 0, x, y );
    p->drawLine( 0, y, x, y );

    //Add the text "Some Text" to the center of all cells:
    p->drawText( 0, 0, (x+1), (y+1), AlignCenter, "Some text" );

    //If this is the current cell, draw an
    //extra frame around it:
```

```
50:     if( (row == curRow) && (col == curCol) )
51:     {
52:         //If we have focus, draw a solid
53:         //rectangle around the current cell:
54:         if( hasFocus() )
55:         {
56:             p->drawRect( 0, 0, x, y );
57:         }
58:         //If we don't have focus, draw dashed
59:         //rectangle instead.
60:         else
61:         {
62:             p->setPen( DotLine );
63:             p->drawRect( 0, 0, x, y );
64:             p->setPen( SolidLine );
65:         }
66:     }
67: }
68:
69: //This function takes care of your mouse clicks:
70: void MyMainWindow::mousePressEvent( QMouseEvent *e )
71: {
72:     int oldRow = curRow;
73:     int oldCol = curCol;
74:
75:     QPoint clickedPos = e->pos();
76:
77:     //Find out which cell were clicked:
78:     curRow = findRow( clickedPos.y() );
79:     curCol = findCol( clickedPos.x() );
80:
81:     //If it wasn't the current cell, update both
82:     //the current cell and the old cell:
83:     if( (curRow != oldRow) || (curCol != oldCol) )
84:     {
85:         updateCell( oldRow, oldCol );
86:         updateCell( curRow, curCol );
87:     }
88: }
89:
90: void main( int argc, char **argv )
91: {
92:     QApplication a(argc, argv);
93:     MyMainWindow w;
94:     a.setMainWidget( &w );
95:     w.show();
96:     a.exec();
97: }
```

在这个例子中，调用 setTableFlags()函数为表添加滚动条，需要将适当的参数传递给函数（第 28 和 29 行）。第 32 行改变表元的颜色。

paintCell()函数增加了一些新的功能。首先，第 46 行为所有表元添加一个简单的文本标签。当然，你可能不希望像例子中那样为 240 个表元添加相同的文本！但是，其方法总是相同的。之后，paintCell()函数检查即将绘制的表元是否为当前所选中的表元。如果程序获得焦点（也就是说，是你当前所选中的程序），并且当前所绘制的表元就是被选中的表元，

那么，将在该表元周围再绘制一个矩形。如果程序没有获得焦点，矩形将为虚线。图 6-8 为该程序执行结果。

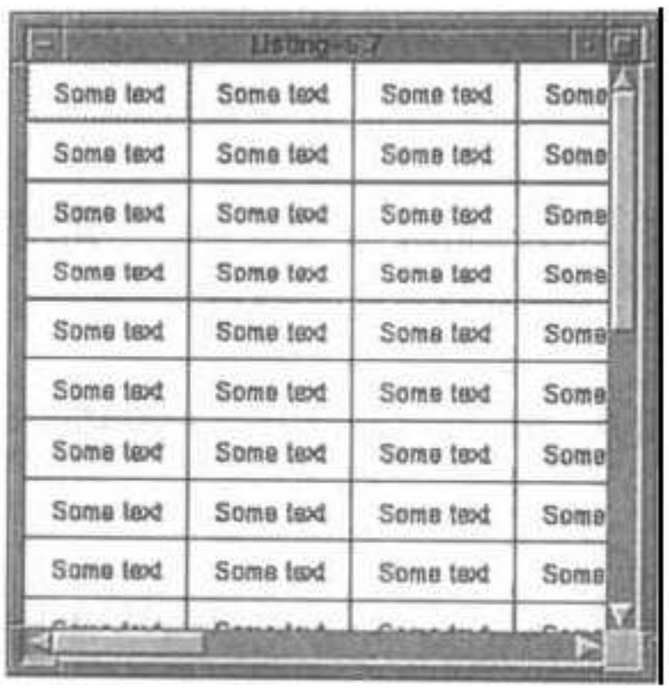


图 6-8　左上角的表元为当前所选中的表元

6.3.3　增加表头

你可能需要向表中行和列增加表头，这由 QHeader 类实现。程序清单 6-8 就是这样一个例子。

程序清单 6-8　　具有水平表头的表

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qtableview.h>
 4: #include <qpainter.h>
 5: #include <qheader.h>
 6: #include <qlabel.h>
 7:
 8: class MyTable : public QTableView
 9: {
10: public:
11:     MyTable( QWidget *parent = 0 );
12: private:
13:     void paintCell( QPainter *, int, int );
14: };
15:
16: MyTable::MyTable( QWidget *parent ) : QTableView( parent )
17: {
18:     setNumCols( 5 );
19:     setNumRows( 5 );
20:     setCellWidth( 100 );
21:     setCellHeight( 30 );
22:     setBackgroundMode( PaletteBase );
23: }
24:
25: void MyTable::paintCell( QPainter* p, int row, int col )
26: {
27:     int x = (cellWidth( col ) - 1);
28:     int y = (cellHeight( row ) - 1);
29:
30:     p->drawLine( x, 0, x, y );
```

```
31:     p->drawLine( 0, y, x, y );
32:
33:     if( col == 0 )
34:     {
35:         p->drawText( 0, 0, (x+1), (y+1), AlignCenter, "Name" );
36:     }
37:     if( col == 1 )
38:     {
39:         p->drawText( 0, 0, (x+1), (y+1), AlignCenter, "Address" );
40:     }
41:     if( col == 2 )
42:     {
43:         p->drawText( 0, 0, (x+1), (y+1), AlignCenter, "City" );
44:     }
45:     if( col == 3 )
46:     {
47:         p->drawText( 0, 0, (x+1), (y+1), AlignCenter, "Gender" );
48:     }
49:     if( col == 4 )
50:     {
51:         p->drawText( 0, 0, (x+1), (y+1), AlignCenter, "Tel." );
52:     }
53: }
54:
55: class MyMainWindow : public QWidget
56: {
57: public:
58:     MyMainWindow();
59: private:
60:     MyTable *table;
61:     QHeader *header;
62:     QLabel *label;
63: };
64:
65: MyMainWindow::MyMainWindow()
66: {
67:     resize( 500, 250 );
68:
69:     table = new MyTable( this );
70:     table->setGeometry( 0, 100, 500, 150 );
71:
72:     header = new QHeader( this );
73:     header->setGeometry( 0, 70, 500, 30 );
74:     header->setOrientation( Horizontal );
75:     header->addLabel( "Name", 100 );
76:     header->addLabel( "Address", 100 );
77:     header->addLabel( "City", 100 );
78:     header->addLabel( "Gender", 100 );
79:     header->addLabel( "Tel.", 100 );
80:
81:     label = new QLabel( this );
82:     label->setGeometry( 0, 0, 500, 70 );
83:     label->setAlignment( AlignCenter );
84:     label->setText( "Let's pretend this is a real
85:             program that needs to present
86:             personal information in a table." );
87: }
88:
```

```
89: void main( int argc, char **argv )
90: {
91:     QApplication a(argc, argv);
92:     MyMainWindow w;
93:     a.setMainWidget( &w );
94:     w.show();
95:     a.exec();
96: }
```

在这个例子中，程序清单 6-7 中的 QMainWindow 类被重新命名为 MyTable。这里创建另外一个类，叫做 MyMainWindow，在其中创建 MyTable 类对象。

这里你第一次看到所创建的新类不是主部件。注意，第 11 行和第 16 行向 MyTable 构造函数添加一些新的内容。在第 11 行，定义构造函数时使用一个 QWidget 对象指针做参数。这个参数代表其父部件。但是，不需要编写处理该参数的代码。它被传递给 QtableView 的构造函数（第 16 行）。这里，传递给 MyTable 构造函数的参数为 this 指针（第 69 行）。这使尚未创建的 MyMainWindow 对象作为其父部件。

第 72 行到 79 行创建表头。这部分代码不需要进一步解释。图 6-9 为该程序的运行结果。

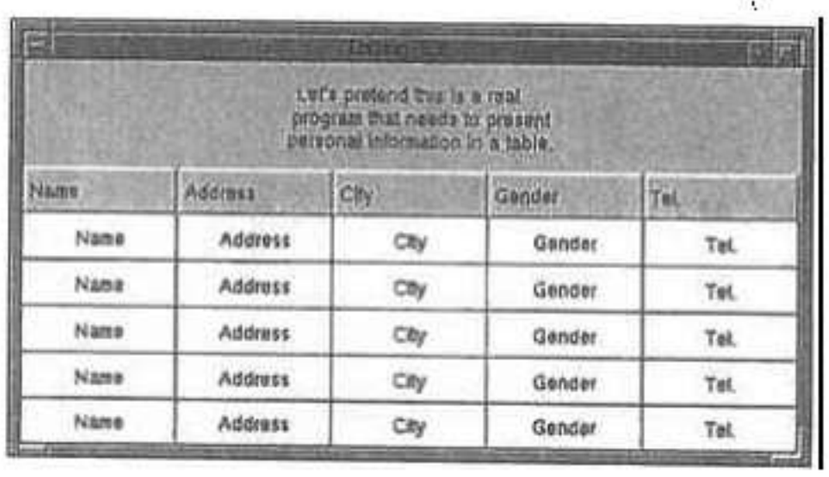


图 6-9　具有水平表头的表

6.4　小　结

这一学时，你学习了怎样使用按钮，包括按钮（Push Button）、单选按钮和复选按钮。按钮可能是程序中最常使用的部件，因此，按钮一节是重要的一节。可以想象，如果 GUI 程序中没有任何按钮，它肯定会逊色不少。

接下来，你学习了怎样实现文本和数字标签，学习了到目前为止标签中最常使用的类 QLabel。也介绍了 QLCDNumber 类，它用于显示数字信息。标签，与按钮一样，都是所有 GUI 库的基本部件。

最后，学习了怎样使用表。这一学时只介绍些基本内容。记住，QTableView 类需要进一步完善。如前所述，它是一个非常抽象的类，因此，还需要改进以使它变得真正有用。但是，这也有其好处，因为它可以使你准确确定应该做哪些事情，以及应该包含哪些功能。

6.5　问题与答案

问：看不到或只能部分看到我向单选按钮和复选按钮所添加的标签。这是为什么？

答：确保留给标签足够的空间。文本也许在其他一些对象的后面。

问：当我使用 QLabel 对象显示一些文本时，为什么会造成其他一些部件（或其中的一部分）的消失？

答：QLabel 对象是一个矩形对象（尽管你看到的可能不是这样），它需要的空间为整个矩形大小，而不仅仅是其文本大小。因此，如果你的标签对象为 400 像素宽，但是，其中间的文本只有 50 像素，该对象仍将覆盖 400 个像素位置。

问：我未看出程序清单 6-7 中的点击选择功能的用途，是不是我做错了什么？

答：不，你没有做错任何事情。包含这个功能只是为了向你显示怎样添加功能。你需要实现其他功能（如让用户修改表元中的内容）才能使该功能变得有用。

6.6 作　业

希望你完成下面的问题和练习，这有助于你牢记本学时所学内容。

6.6.1 测验

1. 不同类型的按钮都有什么作用？
2. 检查是否点击 QPushButton 对象使用什么信号？
3. 布置按钮需要使用哪个类？
4. 哪个函数能够设置 QLabel 对象中文本的对齐方式？
5. 怎样改变 QLabel 对象中文本的大小、字体和样式？
6. 当基于 QTableView 创建子类时，除构造函数外，还有哪个函数需要实现？
7. 为什么需要实现这个函数？
8. 使用哪个类创建表头？

6.6.2 练习

1. 创建一个具有 3 个按钮和 3 个复选按钮的程序。使用按钮控制是否选取复选按钮。
2. 编写一个程序，它具有一个信号部件——一个 QLabel 对象。向它添加文本“*I think Qt is A Good, Fast, Professional, And Nice GUI Library*”（或其他你喜欢的文本）。将对齐方式设为水平中心对齐和垂直下对齐。文本大小为 14 像素。使用 Times 字体，并斜体显示。
3. 制作一个用于显示计算机信息的表。向其添加处理器类型、处理器速度和内存数量列。向这些列添加表头，之后向表添加几项内容。

第 7 学时　认识 Qt 部件的第 2 课

这一学时，将继续学习 Qt 部件。你将看到用于选择的部件、用于安置其他部件的部件，以及用于让用户选择数值的部件。

这一学时所提供的内容不是 GUI 开发中的新内容，但是，这些内容是每个 Qt 程序员都应该理解的非常有用的 Qt 知识，

7.1　选择部件

选择部件使用户能够从预定义的条目菜单中做出选择。这包含列表框和组合框。Microsoft Word 中的字体选择框（组合框）就是一个选择部件的例子。在 Qt 中，QListBox 类用于列表框，QComboBox 类用于组合框。

7.1.1　列表框

QListBox 列表框部件一般用于使用户从中选择一个或多个条目。条目通常为文本类型，但也可以是位图。程序清单 7-1 就是一个例子。

程序清单 7-1　　QListBox 例子

```
#include <qapplication.h>
#include <qwidget.h>
#include <qlistbox.h>

class MyMainWindow : public QWidget
{
public:
        MyMainWindow();
private:
        QListBox *listbox;
};

MyMainWindow::MyMainWindow()
{
        setGeometry( 100, 100, 170, 100 );
        listbox = new QListBox( this );
        listbox->setGeometry( 10, 10, 150, 80 );

        //Start the insertion of items:
        listbox->insertItem( "Item 1" );
```

```
21:         listbox->insertItem( "Item 2" );
22:         listbox->insertItem( "Item 3" ),
23:         listbox->insertItem( "Item 4" );
24:         listbox->insertItem( "Item 5" );
25:         listbox->insertItem( "Item 6" );
26: }
27:
28: void main( int argc, char **argv )
29: {
30:         QApplication a(argc, argv);
31:         MyMainWindow w;
32:         a.setMainWidget( &w );
33:         w.show();
34:         a.exec();
35: }
```

这里，第 10 行用于为列表框部件分配内存。然后在第 16 行创建该列表框，第 17 行设置其几何位置。从第 20 行到第 25 行，使用 QListBox::insertItem()函数向列表框中插入 6 个条目。正如你可能猜到的，该函数的参数表示列表框条目标签。图 7-1 为程序清单 7-1 的执行结果。

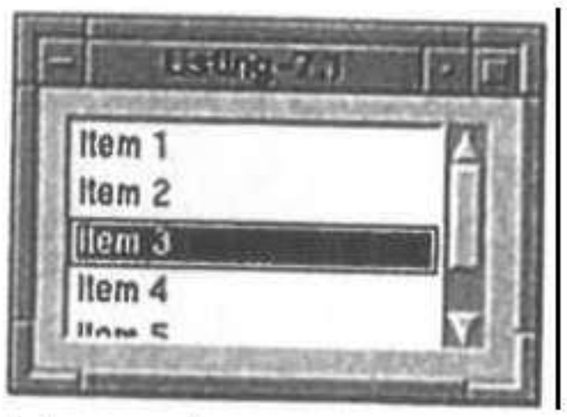


图 7-1　具有 QListBox 对象的窗口

注意，也可以向 insertItem()函数传递一个位图对象，如图 7-2 所示。

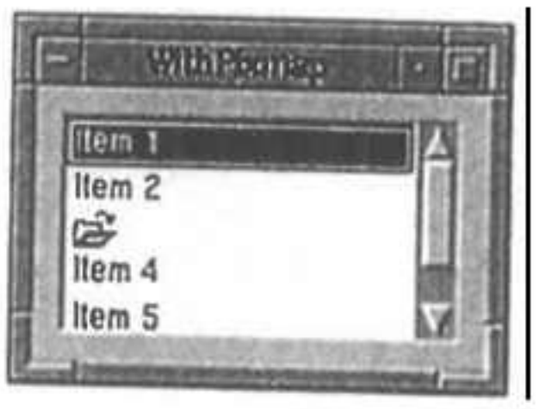


图 7-2　拥有几个文本条目和一个位图条目的 QListBox 对象

当使用 QListBox 对象时，你可用 QListBox::currentItem()函数去检索当前被选中条目的位置。然后在 QListBox::text()或 QListBox::pixmap()中使用这个位置参数去检索当前被选中条目的实际文本或位图。

7.1.2　组合框

如果窗口上没有足够的空间显示一个 QListBox 对象，那么使用 QComboBox 对象代替它将是一个很好的想法。QComboBox 对象的工作方式与 QListBox 非常相似。程序清单 7-2 给出一个这样的例子。

程序清单 7-2　　QComboBox 例子

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qcombobox.h>
 4:
 5: class MyMainWindow : public QWidget
 6: {
 7: public:
 8:         MyMainWindow();
 9: private:
10:         QComboBox *combobox;
11: };
12:
13: MyMainWindow::MyMainWindow()
14: {
15:         setGeometry( 100, 100, 150, 50 );
16:         combobox = new QComboBox( false, this );
17:         combobox->setGeometry( 10, 10, 130, 30 );
18:         //Start the insertion of items:
19:         combobox->insertItem( "Item 1" );
20:         combobox->insertItem( "Item 2" );
21:         combobox->insertItem( "Item 3" ),
22:         combobox->insertItem( "Item 4" );
23:         combobox->insertItem( "Item 5" );
24:         combobox->insertItem( "Item 6" );
25: }
26:
27: void main( int argc, char **argv )
28: {
29:         QApplication a(argc, argv);
30:         MyMainWindow w;
31:         a.setMainWidget( &w );
32:         w.show();
33:         a.exec();
34: }
```

首先，在第 10 行为 QComboBox 对象分配内存。然后在第 16 行执行 QComboBox 的构造函数。QComboBox 构造函数的第 1 个参数决定这个组合框是可读写的，还是只读的。如果它被设置为 false，组合框将为只读组合框。如果它被设置为 true，组合框将变为读写组合框（也就是说，在运行时用户能够动态地向其中插入条目）。与通常一样，第 2 个参数表示这个部件的父对象。第 17 行设置组合框的几何位置。最后，在第 19 行到第 24 行向组合框中插入 6 个条目。这与处理微调框的方法完全一样。图 7-3 为程序清单 7-2 的执行结果。

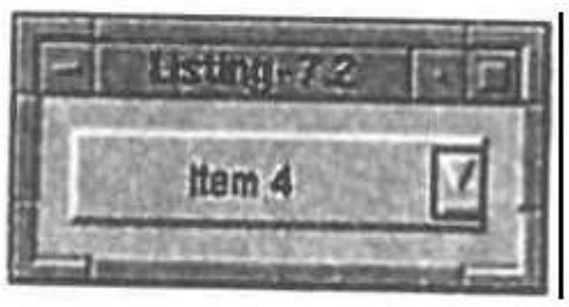


图 7-3　QComboBox 对象，当前选中其第 4 个条目

使用 insert ()函数能够向 QComboBox 对象中插入位图。但是，不能使用位图作为读写组合框的条目。因为，用户不可能去编辑位图。

7.2　部件布局

在程序中合理安排部件会使程序看起来更美观，也使它变得更加容易使用。Qt 提供几个类，它们使部件布局更加简单：

- QGroupBox；
- QButtonGroup；
- QSplitter；
- QWidgetStack。

这一节将逐个介绍这些类。



QFrame 也可用于布局部件，但是，它更是一个用于创建用户类的基类。

7.2.1　QGroupBox 类

QGroupBox 用于在部件周围绘制一个框架。你也可以在框架的上端添加一些描述信息。程序清单 7-3 就是一个例子。

程序清单 7-3　　QGroupBox 例子

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qgroupbox.h>
 4: #include <qlabel.h>
 5:
 6: class MyMainWindow : public QWidget
 7: {
 8: public:
 9:         MyMainWindow();
10: private:
11:         QGroupBox *groupbox;
12:         QLabel *label;
13: };
14:
15: MyMainWindow::MyMainWindow()
16: {
17:         setGeometry( 100, 100, 150, 100 );
18:
19:         groupbox = new QGroupBox( this );
20:         groupbox->setGeometry( 10, 10, 130, 80 );
21:         groupbox->setTitle( "A Group Box" );
22:
23:         label = new QLabel( this );
24:         label->setGeometry( 30, 35, 90, 40 );
25:         label->setText( "Add widgets\n here!" );
26:         label->setAlignment( AlignHCenter | AlignVCenter );
27: }
```

```
28:
29: void main( int argc, char **argv )
30: {
31:         QApplication a(argc, argv);
32:         MyMainWindow w;
33:         a.setMainWidget( &w );
34:         w.show();
35:         a.exec();
36: }
```

这里，第 11 行为 QGroupBox 对象分配内存。然后在第 19 行和第 20 行创建和放置分组框。这里你唯一不熟悉的是 QGroupBox::setTitle()（第 21 行）。你可能会猜想到，它用于设置分组框的标题。该例子执行结果如图 7-4 所示。



图 7-4　QGroupBox 对象和 QLabel 对象窗口

7.2.2　QButtonGroup 类

QButtonGroup 与 QGroupBox 非常类似。但是，它有一些布置按钮方面的特殊功能。QButtonGroup 对象的最常用方法是布置单选按钮。在 QButtonGroup 中放置一套单选按钮能够确保用户一次只选取（点击）一个按钮（使它们之间互相排斥）。第 6 学时“认识 Qt 部件：第 1 课”中的程序清单 6-2 就是这样一个例子。

当将单选按钮插入到 QButtonGroup 对象中时，它们之间自动互相排斥。但是，为了使其他按钮之间互斥，则必须调用 QButtonGroup::setExclusibe(true)函数。

在第 6 学时的“按钮”一节中的例子说明怎样使用这个布局部件。

7.2.3　QSplitter 类

拆分对象使用户能够通过拖动拆分器所提供的部件间的分界线控制子部件的大小。Qt 所提供的 QSplitter 类实现这一功能。程序清单 7-4 给出这样一个例子。在这个例子中，创建一个拆分器，使用户能够改变两个按钮的大小。

程序清单 7-4　　QSplitter 例子

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qsplitter.h>
4: #include <qpushbutton.h>
5:
6: class MyMainWindow : public QWidget
7: {
8: public:
9:         MyMainWindow();
10: private:
11:         QSplitter *splitter;
12:         QPushButton *b1, *b2;
```

```
13: };
14:
15: MyMainWindow::MyMainWindow()
16: {
17:         setGeometry( 100, 100, 150, 100 );
18:
19:         splitter = new QSplitter( this );
20:         splitter->setGeometry( 10, 10, 130, 80 );
21:
22:         b1 = new QPushButton( "Button 1", splitter );
23:         b2 = new QPushButton( "Button 2", splitter );
24: }
25:
26: void main( int argc, char **argv )
27: {
28:         QApplication a(argc, argv);
29:         MyMainWindow w;
30:         a.setMainWidget( &w );
31:         w.show();
32:         a.exec();
33: }
```

这里，第 19 行创建拆分器，第 20 行设置其几何位置。然后，在第 22 行和 23 行创建两个按钮，并使用拆分器作为它们的父对象。这样做，即在两个按钮之间插入一个拆分器。该程序执行结果如图 7-5 和图 7-6 所示。

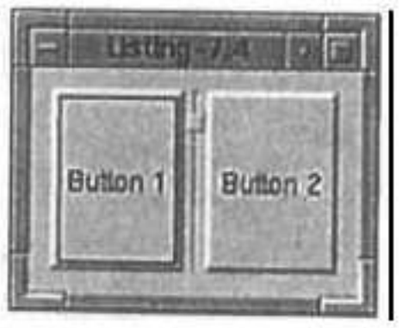


图 7-5　以原始格式显示的 QSplitter 程序

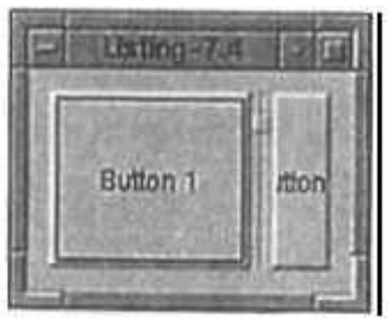


图 7-6　向右拖动拆分器后的显示结果

正如你看到的，当自左向右拖动拆分器的边界时，将使两个按钮一个变大，一个变小。

缺省时，在用户调整完部件尺寸之前（放开边界），QSplitter 不会改变部件大小。但是通过调用 QSplitter::setOpaqueSize(true)函数，能够使部件尺寸随着用户拖动边界而调整。

也可以创建垂直拆分器，这通过调用 QSplitter::setOrientation(Vertical)函数或向构造函数添加一个参数 QSplitter（Vertical, this）来实现。

你可能还需要设置拆分器中部件的最小尺寸。这由 setMinimumSize()函数实现。因此，在前一个例子中，你可以限制用户避免使按钮过小而不能完全显示出其标签。你还可以对 QSplitter 对象使用 setMaximimSize()函数。

最后，你能够定义部件大小是否随着拆分器尺寸的改变而改变。这由

QSplitter::setResizeMode()函数实现。

7.2.4 QWidgetStack 类

当你有多个部件，但每次只希望显示出一个部件时，就需要使用 QWidgetStack 类。程序清单 7-5 给出一个例子。在这个例子中，通过点击两个按钮，你能够控制另外两个按钮中的哪一个是可见的。

程序清单 7-5　　QWidgetStack 例子

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qwidgetstack.h>
 4: #include <qpushbutton.h>
 5:
 6: //Make sure you place the class declaration
 7: //in a header-file of its own, before you run
 8: //MOC on it. Then include the moc file in the
 9: //file holding the class definition.
10:
11: class MyMainWindow : public QWidget
12: {
13:     Q_OBJECT
14: public:
15:     MyMainWindow();
16: private:
17:     QWidgetStack *widgetstack;
18:     QPushButton *b1, *b2, *cb1, *cb2;
19:
20: public slots:
21:     void showb1();
22:     void showb2();
23: };
24:
25: void MyMainWindow::showb1()
26: {
27:     widgetstack->raiseWidget( b1 );
28: }
29:
30: void MyMainWindow::showb2()
31: {
32:     widgetstack->raiseWidget( b2 );
33: }
34:
35: MyMainWindow::MyMainWindow()
36: {
37:     setGeometry( 100, 100, 150, 130 );
38:
39:     widgetstack = new QWidgetStack( this );
40:     widgetstack->setGeometry( 10, 10, 130, 80 );
41:
42:     b1 = new QPushButton( "Button 1", this );
43:     b2 = new QPushButton( "Button 2", this );
44:
```

```
45:     widgetstack->addWidget( b1, 1 );
46:     widgetstack->addWidget( b2, 2 );
47:
48:     cb1 = new QPushButton( "Raise b1", this );
49:     cb1->setGeometry( 10, 100, 60, 20 );
50:
51:     cb2 = new QPushButton( "Raise b2", this );
52:     cb2->setGeometry( 80, 100, 60, 20);
53:
54:     connect( cb1, SIGNAL( clicked() ), this, SLOT( showb1() ) );
55:     connect( cb2, SIGNAL( clicked() ), this, SLOT( showb2() ) );
56: }
57:
58: void main( int argc, char **argv )
59: {
60:     QApplication a(argc, argv);
61:     MyMainWindow w;
62:     a.setMainWidget( &w );
63:     w.show();
64:     a.exec();
65: }
```

在这个程序中定义两个用户槽（第 25 行到第 33 行），这两个槽都非常简单。它们调用 QWidgetStack::raiseWidget()函数使 b1 或 b2 变为可见的（第 27 行和第 32 行）。然后将这些槽连接到 cb1 和 cb2 按钮的 clicked()信号（第 54 和 55 行）。这样做能够控制哪个按钮（b1 或 b2）是可见的。图 7-7 和 7-8 显示出该程序的执行情况。注意，这两个图只是用于说明，实际程序中很少使用这个功能。



图 7-7　程序显示在屏幕上的原始样式

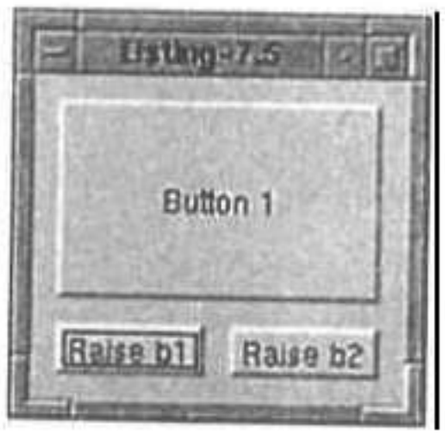


图 7-8　点击 Raise b1 时的显示结果

当然，你可以向部件栈中添加更多的部件。只要记着使用 QWidgetStack::addWidget()函数。正如你在程序清单 7-5（第 45、46 行）中所看到的一样，应该给该函数两个参数：指向被添加部件的指针和该部件在栈中的唯一整数标识号。标识号可以代替部件指针用做 QWidgetStack::raiseWidget()函数的参数。

7.3　滑动框和微调框

这一节将学习怎样使用创建滑块和滚动条的 Qt 类。滑动框使用户能够用鼠标拖动调节器来选择数值。微调框也用于选择数值，但是，它的用法有一点不同（在微调框中，通过点击按钮，而不是拖动调节器，来改变数值）。

7.3.1　QSlider 类

无论创建何种类型的滑动框(Slider)，都要使用 QSlider 类。与所有其他 Qt 类一样，QSlider 类也很容易使用。程序清单 7-6 就是一个例子。

程序清单 7-6　　QSlider 例子

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qslider.h>
 4:
 5: class MyMainWindow : public QWidget
 6: {
 7: public:
 8:         MyMainWindow();
 9: private:
10:         QSlider *slider;
11: };
12:
13: MyMainWindow::MyMainWindow()
14: {
15:         setGeometry( 100, 100, 150, 50 );
16:
17:         slider = new QSlider( 0, 100, 10, 50, Horizontal, this );
18:         slider->setGeometry( 10, 15, 130, 20 );
19:         slider->setTickmarks( QSlider::Below );
20: }
21:
22: void main( int argc, char **argv )
23: {
24:         QApplication a(argc, argv);
25:         MyMainWindow w;
26:         a.setMainWidget( &w );
27:         w.show();
28:         a.exec();
29: }
```

你可能已经注意到，QSlider 类构造函数的参数比较多（第 17 行）。其第 1 个参数设置滑动框的最小值，第 2 个参数设置其最大值。第 3 个参数设置当点击左边或右边的调节标尺时滑块跳动的距离。第 4 个参数设置滑动框的默认值。第 5 个参数设置滑块方向（Horizontal 或 Vertical，即水平方向或垂直方向），最后一个参数为指向滑动框父部件的指针（可能与你猜想的一致）。

第 19 行调用 QSlider::setTickmarks()函数，设置滑动框下面所显示的跳动标记。它们使

用户能够更加清晰地看到滑动框的当前值。图 7-9 为该例子的显示结果。



图 7-9　具有跳动标记的滑动框

但是，如果没有办法告诉你滑动框的当前值，滑动框也就没有太大用处。幸运的是，Qt 提供几个方法使这项工作变得非常简单。这些函数如表 7-1 所示。

表 7-1　　QSlider 成员函数

函　数	说　明
value()	返回滑动框的当前值
sliderMoved(int)	这是一个信号，当滑块的移动量大到能够改变滑动框的当前值时，发射该信号。其参数包含新的值
valueChanged(int)	这是一个信号，当用户结束移动调节器时发射该信号（也就是说，用户选择了一个新值）。参数包含新值
sliderPressed()	每当用户点击调节标尺时均发射该信号
sliderReleased()	用户在点击或拖动调节器时，当释放它们时发射该信号
setValue(int)	设置滑动框值

滑动框有多种用途。稍微想象一下，你就会有一些有趣的想法。你可能已经看到很多程序中都使用滑动框来设置各种数值。只要你认为比微调框更适合的地方都应该使用滑动框，例如，当需要选择一个大范围的数值时。

7.3.2　QSpinBox 类

微调框（Spin Box）使用户通过使用鼠标或键盘来选择数值。与滑块相比较，它占用的空间更少，在某些情况下，也更容易使用。程序清单 7-7 给出一个简单的微调框例子。

程序清单 7-7　　QSpinBox 例子

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qspinbox.h>
4:
5: class MyMainWindow : public QWidget
6: {
7: public:
8:         MyMainWindow();
9: private:
10:        QSpinBox *spinbox;
11: };
12:
13: MyMainWindow::MyMainWindow()
14: {
15:        setGeometry( 100, 100, 150, 50 );
16:
17:        spinbox = new QSpinBox( 0, 100, 5, this );
18:        spinbox->setGeometry( 50, 10, 50, 30 );
19: }
```

```
20:
21: void main( int argc, char **argv )
22: {
23:         QApplication a(argc, argv);
24:         MyMainWindow w;
25:         a.setMainWidget( &w );
26:         w.show();
27:         a.exec();
28: }
```

程序清单 7-7 的显示结果如图 7-10 所示。



图 7-10　一个简单的微调框

QSpinBox 的构造函数有 4 个参数（第 17 行）。第 1、2 个参数分别设置微调框的最小和最大值。第 3 个参数为步长值——也就是每点击一次箭头按钮时微调框数值的改变量。与通常一样，最后一个参数为其父部件。

QSpinBox 也提供了一些有用的成员函数，这些函数如表 7-2 所示。

表 7-2　QSpinBox 成员函数

函　数	说　明
value()	返回微调框的当前值
setValue(int)	设置微调框值
setSuffix()	设置显示在数值后的字符串，例如，日、月、年等之类
setPrefix()	设置显示在数值前的字符串
setSpecialValueText()	为微调框的最小值定义一个替换显示文本，例如，它可能为：*你必须选择一个比此大的值*
setWrapping()	如果已经达到最大或最小值时你再点击箭头按钮将没有任何效果。但是，如果调用 setWrapping(true) 函数，之后当到达最大（或最小）值时再点击箭头，数值将重新从最小（或最大）值开始
stepUp()	模仿鼠标点击上按钮
stepDown()	模仿鼠标点击下按钮

当数值范围不大时，在设置数值时微调框非常有用。相反，如果数值范围很大（如从 1 到 1000），需要点击的次数就会很多！在这种情况下，使用滑块是比较好的选择。然而，如前所述，如果所设置的数值范围较小（如设置当前日期），微调框则是最好的选择！

7.4　小　结

这一学时，你学习了怎样使用选择部件。使用选择部件（对于文本字符串和数值）时，你需要了解已提供的类方法。在窗口中布置选择部件并不困难，但是，你很可能想使程序中的这些选择部件具有一定的功能。

如果你不知道某个事件发生时所发射的信号，或者不知道应该调用哪个函数去改变部

件，这都将导致你无法实现部件的功能。尽管不可能完全记住这些信息，但是你应该对这一学时所学习的选择部件的工作机制有一个基本理解。此后，当你需要进一步了解某个方法时，可以查看 Qt Reference Document。

这一学时也学习了布局部件。布局部件是用于布置其他部件的部件。尽管需要在程序中插入一些额外代码才能实现布局部件的功能，但是，你还是应该尽可能地使用它们。如果正确使用布局部件，将会使应用程序界面更加美观，也使程序更加容易使用。

7.5　问题与答案

问：我使用这一学时中所介绍的部件时，看不到或只能部分看到部件。这是什么原因？

答：这是 Qt 初级程序员中存在的一个普遍问题。然而其解决方法与问题本身一样简单。你知道，所有的 Qt 部件形状均为矩形，尽管它们通常显示出来可能不是矩形。因此，一个部件将占用的空间比它们看起来所占用的空间要多。当一个或多个部件未显示在屏幕上，或是它们部分被隐藏时，问题很可能是这些部件被其他部件（那些看起来所占用空间比它们实际占用空间要少的部件）所覆盖。

问：当我编译程序清单 7-5 时出现很多错误，这是为什么？

答：你必须将这段代码分为一个.cpp 文件和一个 h-文件（头文件）。你还必须使用元对象编译器，这样做才能使用户槽工作。有关槽的详细说明请看第 4 学时“槽和信号”。

问：什么时候需要使用 QSpinBox::setSpecialValueText()函数？你能举一个简单的例子吗？

答：如果你想让用户在一个新的字处理软件中改变字体大小，QSpinBox::setSpecialValueText()函数就非常有用。你知道，字体大小不能设置为 0。然而，用户很可能没有想到这一点，并试图将字体大小设置为最小值。这时，如果不使用 QSpinBox::setSpecialValueText()函数，用户将把字体大小设置为 0，这将导致很离奇的问题。相反，如果使用 QSpinBox::setSpecialValueText()函数，并将文本设置为*必须大于 1* 之类的字符串，问题就不会发生。

7.6　作　业

完成这一节中的测验和练习将使你理解本学时所学所有内容。

7.6.1　测验

1. 什么是选择部件？
2. Qt 提供哪两种类型的文本选择部件？
3. QSplitter 类的用途是什么？
4. QWidgetStack 类的用途是什么？
5. 当使用部件栈时，什么时候应该使用整数标识号代替部件指针？
6. 尽管没有明确指出这一学时中的选择部件，但你学习了两种用于数值的选择部件，

你能指出它们的名字吗？

7. 与微调框相比，滑动框的优点是什么？

7.6.2　练习

1. 编写一个简单的程序，它包含列表框和组合框。并使用 for 循环向列表框或组合框中插入几个条目。
2. 实验 QSplitter 类。是否能够向拆分器中插入其他布局部件？能够插入具有子部件的部件吗？
3. 编写拥有一个滑块和一个微调框的应用程序。连接两个部件，使你在拖动滑块时微调框的值能够随之改变。

第 8 学时　认识 Qt 部件的第 3 课

这是这一部分的最后一课。在这一学时，你将学习几个新的部件类型。这一学时依旧无法介绍类的所有内容和所提到的所有函数。这就是为什么说在学习 Qt 过程中 Qt Reference Document 非常重要的原因（也就是作为一个参考）。

这一学时你将学习使用 3 种新的部件：文本输入域、列表视图和进程条。文本输入域用于处理用户的文本输入操作。列表视图是一个非常有趣的部件，它能够有效地显示数据。进程条则用于将程序所执行任务的状态信息通知用户，这些部件是基本部件，因此，在程序中你很可能会（迟早）需要它们。

8.1　文本输入域

文本输入域是一种在其中能够输入文本的部件。例如，在一个文本编辑器中，能够读写文本的白色区域就是文本输入域。Qt 提供两个用于创建文本输入域的类：QLineEdit 和 QMultiLineEdit。二者都将在这一节中介绍。

8.1.1　QLineEdit

QLineEdit 创建单行文本输入域。它能够（也应该）用于读取用户输入的短字符串，如用户名和口令等。程序清单 8-1 显示一个 QLineEdit 对象。

程序清单 8-1　　QLineEdit 例子

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qlineedit.h>
4:
5: class MyMainWindow : public QWidget
6: {
7: public:
8:         MyMainWindow();
9: private:
10:        QLineEdit *edit;
11: };
12:
13: MyMainWindow::MyMainWindow()
14: {
```

```
15:         setGeometry( 100, 100, 200, 50 );
16:
17:         edit = new QLineEdit( this );
18:         edit->setGeometry( 10, 10, 180, 30 );
19: }
20:
21: void main( int argc, char **argv )
22: {
23:         QApplication a(argc, argv);
24:         MyMainWindow w;
25:         a.setMainWidget( &w );
26:         w.show();
27:         a.exec();
28: }
```

然而，程序清单 8-1 没有提供任何真正新的内容。这只是像通常一样创建和放置部件。但是，它显示了新部件 QLineEdit 的用途。图 8-1 显示出该程序的运行结果。

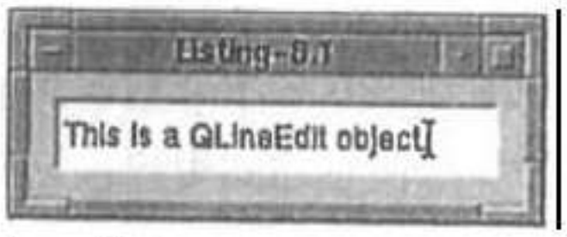


图 8-1 具有 QLineEdit 对象的简单程序

使用 QLineEdit::setText()函数能够手工设置 QLineEdit 对象中的文本。如果想检索当前写入到 QLineEdit 对象中的文本，则可调用 QLineEdit::text()函数。当文本改变时，将发射 QLineEdit::valueChanged()信号。

QLineEdit 的一个普通用法是读取口令。因此，QLineEdit 类提供一个函数，它使用户输入到域中的文本显示为*。这是隐藏口令的常用方法，你可能在其他很多程序中已经看到。将下面两行添加到程序清单 8-1 中的构造函数内，将使文本显示为*，并将输入文本的最大长度设置为 8 个字符：

```
edit->setEchoMode( QLineEdit::Password ) ;
edit->setMaxLength( 8 );
```

现在，你在域中所输入的文本将显示为图 8-2 所示的样子。

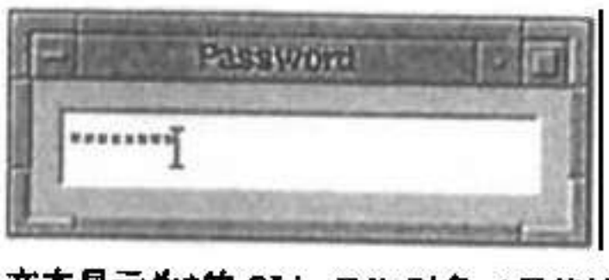


图 8-2 将输入文本显示为*的 QLineEdit 对象（只能输入 8 个字符）

注意，当你输入 8 个字符后，不再显示更多的*。这意味着不能再输入其他字符。

当使用 QLineEdit 对象时，你应该了解 returnPressed()和 textChanged()信号。当按下回车键时，将发射 returnPressed()信号。当文本改变时将发射 textChanged()信号。textChanged()也包括新的文本。

8.1.2 QMultiLineEdit

QMultiLineEdit 是 Qt 提供的另一个文本输入域。在本书前面的一个例子中已经使用过

它。你或许还能够记得，QMultiLineEdit 提供一个多行矩形区域，你可以在其中显示文本或让用户输入文本。

QMultiLineEdit 的使用方法与 QLineEdit 基本相同。但是，QMultiLineEdit 包含几个新的用于编辑文本的函数。使用 insertAt()函数，能够在 QMultiLineEdit 对象中的某一位置插入文本。insertAt()函数具有 3 个参数：第 1 个参数是需要插入的字符串，第 2 个参数指出所插入文本的行号，第 3 个参数指出在第几个字符之后插入文本。因此，下面代码将在第 12 行的第 37 个字符之后插入 Hey!：

```
object->insertAt( "Hey!", 12, 37 ) ;
```

也可以使用 QMultiLineEdit::insertLine()函数插入文本，这是一个稍微简单一些的函数，它只带一个或两个参数。例如：

```
object->insertLine( "Hey!", 5 ) ;
```

这行代码将在第 5 行插入 Hey!。但是，如果省略第 2 个参数，文本将被添加到当前最后一行下的一个新行上。

调用 QMultiLineEdit::removeLine()函数能够删除一行文本。其参数为待删除行的行号。



QLineEdit 和 QMultiLineEdit 都能够使用系统剪贴板。这通过调用 cut()、copy()和 paste()函数实现。

更多信息请参考 Qt Reference Document。这时，你将发现 Qt Reference Document 很有帮助，因为 QMultiLineEdit 包含很多有用的函数。这些函数包括用于使 QMultiLineEdit 对象变为只读的 setReadOnly()函数（当只想显示文本时，该函数非常有用），检索 QMultiLineEdit 对象中当前文本的 text()函数，以及清除其中所有文本的 clear()等。

8.2　理解列表视图

列表视图是我个人最喜爱的部件之一。这个部件所显示的数据非常美观，并且很容易查找。尽管列表视图是一个非常复杂的部件，但是 Qt 使这一部件的实现变得非常简单。你可能已经猜想到，该类的名称为 QListView。

程序清单 8-2 是一个简单的 QListView 使用例子。

程序清单 8-2　　QListView 例子

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qlistview.h>
 4:
 5: class MyMainWindow : public QWidget
 6: {
 7: public:
 8:     MyMainWindow();
 9: private:
10:     QListView *listview;
```

```
11:     QListViewItem *topic1;
12:     QListViewItem *topic2;
13:     QListViewItem *topic3;
14:     QListViewItem *item;
15: };
16:
17: MyMainWindow::MyMainWindow()
18: {
19:     setGeometry( 100, 100, 300, 300 );
20:
21:     //Create the list view:
22:     listview = new QListView( this );
23:     listview->setGeometry( 0, 0, 400, 400 );
24:     //Make the down-arrow visible:
25:     listview->setRootIsDecorated( true );
26:     //Add three columns:
27:     listview->addColumn( "Book" );
28:     listview->addColumn( "Sold Copies" );
29:     listview->addColumn( "Price" );
30:
31:     //Add three items:
32:     topic1 = new QListViewItem( listview, "Topic 1" );
33:     topic2 = new QListViewItem( listview, "Topic 2" );
34:     topic3 = new QListViewItem( listview, "Topic 3" );
35:
36:     //Add three sub-items to each item:
37:     item = new QListViewItem( topic1, "Book 1", "21,000", "$29.99" );
38:     item = new QListViewItem( topic1, "Book 2", "19,000", "$24.99" );
39:     item = new QListViewItem( topic1, "Book 3", "14,000", "$39.99" );
40:
41:     item = new QListViewItem( topic2, "Book 4", "38,000", "$34.99" );
42:     item = new QListViewItem( topic2, "Book 5", "16,000", "$19.99" );
43:     item = new QListViewItem( topic2, "Book 6", "9,000", "$29.99" );
44:
45:     item = new QListViewItem( topic3, "Book 7", "32,000", "$39.99" );
46:     item = new QListViewItem( topic3, "Book 8", "25,000", "$37.99" );
47:     item = new QListViewItem( topic3, "Book 9", "13,000", "$44,99" );
48: }
49:
50: void main( int argc, char **argv )
51: {
52:     QApplication a(argc, argv);
53:     MyMainWindow w;
54:     a.setMainWidget( &w );
55:     w.show();
56:     a.exec();
57: }
```

条目添加过程与菜单创建过程非常类似。首先，在第 22 行到第 29 行创建实际的列表视图。第 25 行调用 QListView::setRootIsDecorated(TRUE)函数，它在顶级条目的左边显示一个箭头，用户使用它能够显示出子条目。之后，在第 27、28 和 29 行使用 QListView::addColumn()函数创建 3 个列。传递给函数的参数为列名。第 32、33 和 34 行创建 3 个顶级条目。这由 QListViewItem 类实现。QListViewItem 类构造函数具有两个参数：第 1 个参数为 QListView 对象，它说明向哪个列表视图中插入这些条目。第 2 个参数表示条目名称。在第 37 到 47 行向每个顶级条目中插入 3 个子条目。这里，向 QListViewItem 构造

函数添加另外两个参数。这两个参数表示子条目的第 2、3 列。该程序的执行结果如图 8-3 所示。

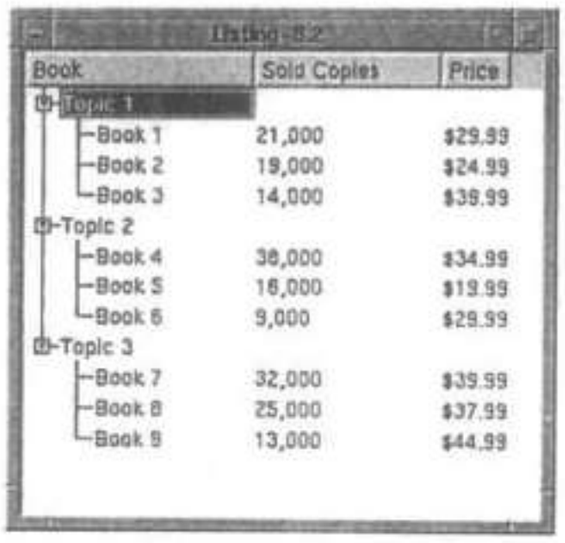


图 8-3　列表视图



注意，不必调用 insertItem()之类的函数，而是依赖于它们间的父子结构。



如果点击列标题，将按该列对列表视图排序。

QListView 还提供其他很多在前面例子中没有使用过的函数。表 8-1 列出其中的一部分。为了能够完全控制和利用 QListView 类，你需要学习使用这些函数。尽管在一个项目中你可能不会需要所有这些函数，但是，知道这些函数的存在将有助于你做出更好的选择。

表 8-1　QListView 成员函数

函　数	说　明
setColumnsWidthMode()	这个方法决定怎样设置列宽度。它带两个参数，第一个为整数，代表所设置列。第二个为枚举值：Maximum 或 Manual。如果设置为 Maximum，列宽度将等于其中最宽条目的宽度。如果设置为 Manual，列宽度将由 setColumnWidth()函数设置
setMultiSelection()	决定用户一次能够选择一个条目还是多个条目。该函数参数为布尔类型（TRUE 表示多选，FALSE 表示单选）
setAllColumnsShowFocus()	该函数参数为 TRUE 或 FALSE。函数被设置为 TRUE 时，所有列将被显示为焦点。测试一下，你将会看到其显示结果
setTreeStepSize()	传递给该函数的整数参数决定一个条目与其父条目之间的相对偏移像素数
setSorting()	该函数决定怎样对列表视图进行排序。它带两个参数：一个整数指出排序列和一个布尔值（TRUE 或 FALSE）。如果将其值设置为 TRUE，将按升序方式进行排序；如果设置为 FALSE，则按降序方式排序
itemAt()	这个函数带一个 QPoint 对象指针做参数。它返回指向指定位置的 QListItem 对象指针
firstChild()	返回列表视图中的第一个条目
itemBelow()	返回列表视图中紧接着该条目的下一个条目
itemAbove()	返回列表视图中紧接着该条目的上一个条目
setSelected()	该函数决定是否选中一个条目。它有两个参数：第一个参数为指向 QListView 对象的指针，第二个参数为布尔值（TRUE 或 FALSE）。为 TRUE 时选中条目
setCurrentItem()	设置键盘输入焦点，使它指向函数指针参数所指向的条目
currentItem()	返回指向当前选中条目的指针
selectionChanged()	该函数是一个信号，当选择改变时（也就是说，如果你选中其他条目，或取消当前所选条目）将发射该信号。它传递一个指向新选条目的指针

续表

函　数	说　明
currentChanged()	当当前条目改变时将发射该信号　它也传递一个指向新条目的指针
doubleClicked()	当用户双击一个条目时发射该信号　所传递的指针指向被双击的 QListView 对象
returnPressed()	当用户输入回车键时发射该信号　它传递的指针指向键盘焦点条目
rightButtonClicked()	当释放鼠标右键时（在它被按下之后）将发射该信号　该信号向槽传递 3 个参数：一个指向被点击条目的指针（可以为 0），点击位置坐标和一个代表被影响列的整数值



学习这个表之后，你可能很想了解当前条目和选中条目之间的区别。当前条目每次只有一个，它是当前焦点所在条目。但是，如果你调用 setMultiSelection(TRUE)函数打开多选选项，则同时可以选中多个条目。在这种情况下，则有一个当前条目和多个选中条目。当前条目是选中条目之一，一个选中条目可以成为当前条目，但当前条目只能是一个选中条目。

现在，来看一个例子，它进一步开发程序清单 8-2 中的程序功能。假设你想实现一个新的功能：当用户点击列表视图中的一个条目时，则弹出该书的描述信息。为了实现这一功能，需要创建一个新的槽，并将它连接到 doubleClicked()信号。程序清单 8-3 实现这一功能。在这个例子中，实现 QMyMainWindow::ShowDscription()槽。每当双击一个条目时，该槽显示一个信息窗口。信息窗口由 QMessageBox 类创建，在第 10 学时“理解 Qt 对话框”中将介绍该类。

程序清单 8-3　　具有消息功能信息列表视图

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qlistview.h>
 4: #include <qmessagebox.h>
 5:
 6: //Remember to make a separate .h file
 7: //of the class declaration, and then
 8: //use the Meta Object Compiler on it.
 9: class MyMainWindow : public QWidget
10: {
11:   Q_OBJECT
12: public:
13:         MyMainWindow();
14: private:
15:         QListView *listview;
16:         QListViewItem *topic1;
17:         QListViewItem *topic2;
18:         QListViewItem *topic3;
19:         QListViewItem *item;
20:     QMessageBox *box;
21: public slots:
22:     void ShowDescription();
23: };
24:
25: //Here is our new slot:
```

```
26: void MyMainWindow::ShowDescription()
27: {
28:     box = new QMessageBox( "Book Info", "Here, we could show some short
29:                 information text about the book you just
30:                 double-clicked on.",
31:                 QMessageBox::Information, QMessageBox::Ok, 0, 0 );
32:     box->show();
33: }
34:
35: MyMainWindow::MyMainWindow()
36: {
37:         setGeometry( 100, 100, 300, 300 );
38:
39:         listview = new QListView( this );
40:         listview->setGeometry( 0, 0, 400, 400 );
41:         listview->setRootIsDecorated( true );
42:
43:         listview->addColumn( "Book" );
44:         listview->addColumn( "Sold Copies" );
45:         listview->addColumn( "Price" );
46:
47:         topic1 = new QListViewItem( listview, "Topic 1" );
48:         topic2 = new QListViewItem( listview, "Topic 2" );
49:         topic3 = new QListViewItem( listview, "Topic 3" );
50:
51:     item = new QListViewItem( topic1, "Book 1", "21,000", "$29.99" );
52:     item = new QListViewItem( topic1, "Book 2", "19,000", "$24.99" );
53:     item = new QListViewItem( topic1, "Book 3", "14,000", "$39.99" );
54:     item = new QListViewItem( topic2, "Book 4", "38,000", "$34.99" );
55:     item = new QListViewItem( topic2, "Book 5", "16,000", "$19.99" );
56:     item = new QListViewItem( topic2, "Book 6", "9,000", "$29.99" );
57:     item = new QListViewItem( topic3, "Book 7", "32,000", "$39.99" );
58:     item = new QListViewItem( topic3, "Book 8", "25,000", "$37.99" );
59:     item = new QListViewItem( topic3, "Book 9", "13,000", "$44.99" );
60:
61:     connect( listview, SIGNAL( doubleClicked( QListViewItem * ) ),
62:         this, SLOT( ShowDescription() ) );
63: }
64:
65: void main( int argc, char **argv )
66: {
67:         QApplication a(argc, argv);
68:         MyMainWindow w;
69:         a.setMainWidget( &w );
70:         w.show();
71:         a.exec();
72: }
```

在第 22 行声明 ShowDescription()槽。从第 26 行到第 33 行实现该槽。正如你所看到的，QMessageBox 类用于（第 28 行到 31 行）创建消息窗口，当用户双击一个条目时弹出该窗口。这是一个 Qt 对话框类，第 10 学时将介绍它。图 8-4 为该程序执行结果。

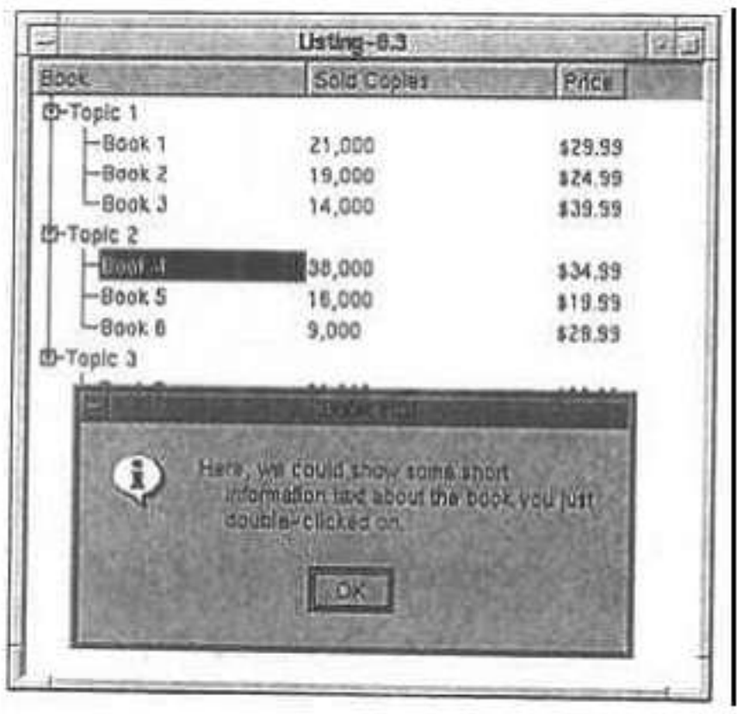


图 8-4　点击“Book 4”条目显示出一个消息框，点击其中的 OK 按钮将关闭它

这一节的内容应该能够使你学会在程序中怎样使用列表视图。

8.3　进程条

进程条这种部件用于向用户显示程序的当前状态。例如，在执行一个非常耗时的任务时，如果你希望向用户提示该任务需要多长时间才能够完成，这时显示一个进程条将是非常有用的。

Qt 提供的 QProgressBar 类用于创建进程条。（另一个类，QProgressDialog，也可用于这一任务。在第 10 学时中将讨论 QProgressDialog）。创建和显示 QProgressBar 的方法与其他 Qt 类一样。但是，为了使进程条担当起进程指示功能，还需要执行其他几项操作（例如，需要定义进程条的刷新时间等）。程序清单 8-4 给出一个简单例子。

程序清单 8-4　　使用滑动框控制进程条

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qslider.h>
 4: #include <qprogressbar.h>
 5:
 6: class MyMainWindow : public QWidget
 7: {
 8: public:
 9:         MyMainWindow();
10: private:
11:     QProgressBar *bar;
12:     QSlider *slider;
13: };
14:
15: MyMainWindow::MyMainWindow()
16: {
17:       setGeometry( 100, 100, 200, 90 );
18:
19:     bar = new QProgressBar( 100, this );
20:     bar->setGeometry( 10, 10, 180, 30 );
21:
```

```
22:     slider = new QSlider( 0, 100, 10, 0, Horizontal, this );
23:     slider->setGeometry( 10, 50, 180, 30 );
24:
25:     //Connect the progress bar and the slider so
26:     //that the slider controls the progress of the
27:     //progress bar:
28:     connect( slider, SIGNAL( valueChanged(int) ),
29:          bar, SLOT( setProgress(int) ) );
30: }
31:
32: void main( int argc, char **argv )
33: {
34:         QApplication a(argc, argv);
35:         MyMainWindow w;
36:         a.setMainWidget( &w );
37:         w.show();
38:         a.exec();
39: }
```

在这个例子中，进程条的 setProgress()槽被连接到滑动框的 valueChanged()信号。这样做使你能够通过拖动滑块来控制进程条的进程。注意，这不是一个实际的例子。但是，它与在真正程序中实现这一功能的概念是相同的。当到达某一点时，只要确保使用适当的参数调用 setProgress()函数即可。



第 19 行中传递给 QProgressBar 构造函数的第一个参数表示进程条达到 100%所需移动的步数。例如，如果你想拷贝 5 个文件，并需要进程条在每拷贝一个文件后移动一步，该值就应该设置为 5。

这个程序的执行结果如图 8-5 和图 8-6 所示。在图 8-5 中，没有拖动滑块，因此，进程条也没有前进。在图 8-6 中，滑块已被向右拖动，因此，进程条也发生变化。

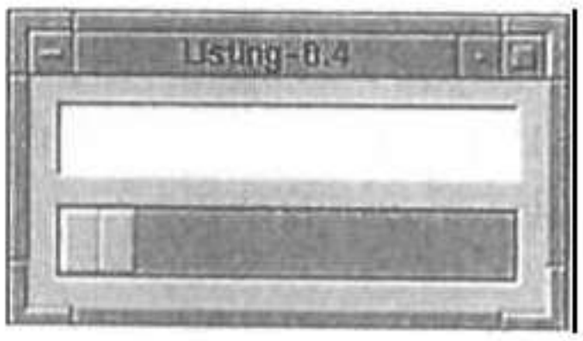


图 8-5　进程条程序显示在屏幕上的初始状态

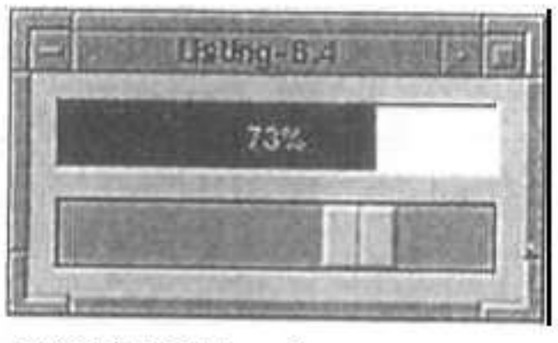


图 8-6　滑块向右拖动时程序的显示结果，最后传递给 setProgress()函数的参数值为 73

进程条虽简单但很有用。如果程序需要执行一个很费时的任务，则需要使用进程条将所发生的事情以及任务还需要多长时间才能完成等信息通知用户。

8.4　小　结

文本输入域和进程条都是非常直观的部件，你应该清楚了解它们的用途。相反，列表视图则完全不同。QlistView 功能强大，但也复杂。你需要深入研究才能完全控制它——通过控制它，你才能够发现一些全新的列表视图使用方法。QlistView 是一个令人激动的类（如果你认为 C++类是令人激动的话），尽管刚学起来难于理解。

你已经学习了 Qt 部件，学习了怎样使用 Qt 所有部件。通过这部分的学习，你已经理解 Qt 部件的工作机制和它们的用途。Qt 部件是一套完整的图形对象集合，足以构造所有种类的 GUI 应用程序。

8.5　问题与答案

问：在程序清单 8-3 中，为什么需要为 QmessageBox 对象调用 show()，而对于其他对象则不需要？

答：这是因为 QmessageBox 对象（框）不是 MyMainWindow 类的子部件。因此，在 main() 中调用 MyMainWindow::show()不会影响 QmessageBox 框。

问：当我编译程序清单 8-3 时编译器显示出很多错误，这是为什么？

答：你必须将这段代码分为一个.cpp 文件和一个.h 文件（头文件）。你还必须对.h 文件使用元对象编译器，并在.cpp 文件中包含 MOC 的输出文件。只有这样做才能使用户槽工作。更详细说明请参看第 4 学时“槽和信号”。

8.6　作　业

与通常一样，你应该完成下面的问题和测验。这将能够使你确实理解本学时所学内容。更重要的是，你也能够学习一些以前不了解的知识。

8.6.1　测验

1. QLineEdit 和 QMultiLineEdit 间的区别是什么？
2. 向用户显示文本文件时应该使用哪个类？
3. 如果只考虑成员函数，那么，QLineEdit 和 QMultiLineEdit 间最本质的区别是什么？
4. 如果不使用 insertItem()函数向列表视图中插入条目，那应该使用哪个方法？
5. 当双击列表视图中的条目时将发射哪个信号？
6. 当在列表视图中按回车键时将发射哪个信号？
7. 什么是进程条？
8. 在应用程序中，什么时候应该使用进程条？

8.6.2 练习

1. 在 Qt Reference Document 中查找能够使 QMultiLineEdit 对象变为只读的 QMultiLineEdit 成员函数。该函数的名称是什么？怎样调用它才能将 QMultiLineEdit 对象设置为只读？
2. 创建一个具有列表视图的程序。该列表视图从左至右包含 3 列——Country、Capital 和 Population。然后为每个洲创建一个顶级条目，并添加一些子条目描述每个洲中的几个国家，包括首都和人口等信息。
3. 在使用进程条时你可能发现它难于实现。也许不清楚什么时候应该调用 setProgress() 函数。一个解决方法（尽管它可能不是最好的方法）就是首先测量完成任务所需时间，之后再创建一个 for 循环（或类似语句），使用它算出该任务的合适时间量。例如，如果执行一个时间长度大约为 30 秒的任务，用一个 for 循环在适当的时间调用 setProgress()函数。

第 9 学时　创建简单图形

当第一次创建新的 GUI 应用程序时，你可能还认识不到这个程序是由大量的低级图形代码组成的。幸运的是，这个代码由 GUI 库产生（这里为 Qt），因此，不必担心它。

你可能想了解低级图形代码是什么。这种代码在屏幕上绘制程序。当创建 Qt 部件时，Qt 使用系统的低级函数（绘制线、矩形等图形的函数）在屏幕上绘制部件。

前面已经提到，Qt 使用低级代码创建 Qt 预定义部件。但是，有时可能需要创建 Qt 没有创建的用户图形对象。在这种情况下，需要某个能够产生用户图形的低级图形产生器。为此，Qt 提供了 QPainter 类。

这一课，将学习怎样使用 QPainter，也将学习怎样在程序中实现打印功能。

9.1　QPainter 类

前面已经介绍，QPainter 类用于在 Qt 中创建用户图形。它非常直观，也不难使用。在 QPainter 类这一节将首先学习怎样创建 QPainter 对象和绘制简单的矩形。

9.1.1　QPainter

为了使用 QPainter 创建用户图形，需要采取一些措施避免干扰 Qt 自己的绘图功能。需要采取的措施包括：将绘图代码放置在一个称做 drawEvent()的特殊函数内，并且通过调用 QPainter::begin()和 QPainter::end()函数来启动和停止绘图。程序清单 9-1 是一个简单的例子。

程序清单 9-1　　绘制简单矩形

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qpainter.h>
 4:
 5: class MyMainWindow : public QWidget
 6: {
 7: public:
 8:         MyMainWindow();
 9: private:
10:         void paintEvent( QPaintEvent* );
11:         QPainter *paint;
12: };
```

```
13:
14: //Here is our implementation of paintEvent():
15: void MyMainWindow::paintEvent( QPaintEvent* )
16: {
17:         paint = new QPainter;
18:         //Start the painting:
19:         paint->begin( this );
20:         //Draw a rectangle:
21:         paint->drawRect( 20, 20, 160, 160 );
22:         //End the painting:
23:         paint->end();
24: }
25:
26: MyMainWindow::MyMainWindow()
27: {
28:         setGeometry( 100, 100, 200, 200 );
29: }
30:
31: void main( int argc, char **argv )
32: {
33:         QApplication a(argc, argv);
34:         MyMainWindow w;
35:         a.setMainWidget( &w );
36:         w.show();
37:         a.exec();
38: }
```

首先在第 17 行创建 QPainter 对象。之后，在第 19 行调用 QPainter::begin()函数开始绘图操作。第 21 行绘制一个矩形。这由 QPainter::drawRect()函数实现。这个函数需要 4 个参数：头两个参数定义矩形左上角位置（相对于窗口），后两个参数定义矩形的宽度和高度。最后，在第 23 行调用 QPainter::end()函数结束绘图。这个程序创建一个 200×200 像素的窗口，在其中绘制一个 160×160 像素的黑色、未填充矩形（如图 9-1 所示）。



图 9-1　具有矩形图形的窗口

正如你所看到的，程序清单 9-1 的结果是绘制一个简单的黑色矩形。这个例子一点也不奇特，但它介绍了 QPainter 类，这正是这一节的目的。

9.1.2　设置绘图样式

在开始实际绘图之前，需要选择 QPainter 绘制线、矩形等图形所使用的样式。这由 QPainter 的几个成员函数实现。

1．选择画笔

使用适当的参数调用 QPainter::setPen()函数，就能够选择所使用的画笔样式。下面我们

来修改程序清单 9-1，使之使用一个 4 像素宽的蓝色画笔绘制虚线。只需将下面一行代码添加到程序清单 9-1 中的 begin()（程序清单 9-1 中的第 19 行）和 end()（程序清单 9-1 中的第 23 行）调用之间：

```
paint->setPen( QPen( blue, 4, QPen::DashLine ) ) ;
```

其结果如图 9-2 所示。

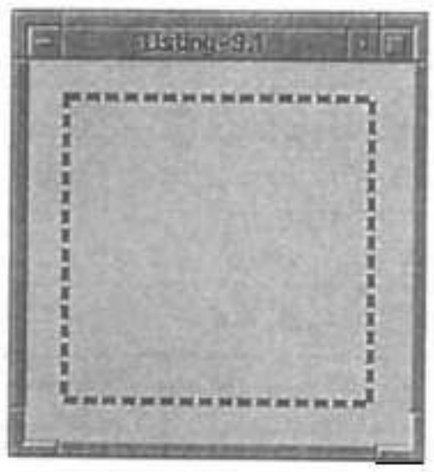


图 9-2　使用 QPaint::setPen()函数改变画笔类型

像你看到的，只需要使用单行代码即可戏剧般地改变绘图样式。

2. 选择填充样式

可以使用图案或一种纯色填充矩形。这由“画刷”（brush）实现。例如，如果想使用纯红颜色填充程序清单 9-1 中的矩形，需将下面一行代码添加到程序清单 9-1 中的 begin()（程序清单 9-1 中的第 19 行）和 end()（程序清单 9-1 中的第 23 行）函数调用之间：

```
paint->setBrush( QBrush( red, SolidPattern ) ) ;
```

这一函数调用结果如图 9-3 所示。

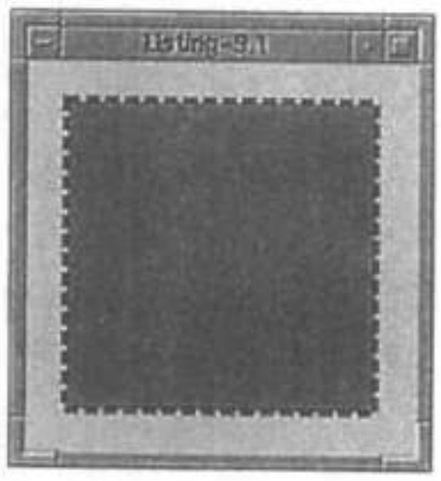


图 9-3　将填充样式设置为纯红颜色

然而，通过改变 SolidPattern 参数，则能够用其他方式填充矩形。例如，如果将它修改为 Dense6Pattern，填充样式将从 100%（SolidPattern）改变为 12%（Dense6Pattern）。图 9-4 为该例子执行结果。

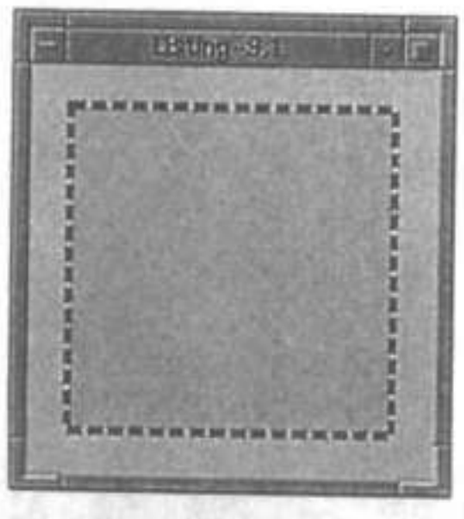


图 9-4　将填充样式从 100%改变为 12%

所有不同填充样式如表 9-1 所示。

表 9-1　　填充样式

填 充 样 式	说　　明
SolidPattern	纯色填充（见图 9-3）
Dense1Pattern	94%填充模式
Dense2Pattern	88%填充模式
Dense3Pattern	63%填充模式
Dense4Pattern	50%填充模式
Dense5Pattern	37%填充模式
Dense6Pattern	12%填充模式（见图 9-4）
Dense7Pattern	6%填充模式
HorPattern	用水平线填充
VerPattern	用垂直线填充
CrossPattern	使用交叉线（网格图案）填充
BDiagPattern	对角线填充，它们指向右上角
FDiagPattern	对角线填充，它们指向左上角
DiagCrossPattern	对角线相互交叉方式
CustomPattern	使用位图图案填充

3．改变字体

使用 QPainter 也能够绘制文本。这样做时，很可能需要改变字体。这非常简单——只需要调用 setFont()函数。程序清单 9-2 给出一个处理文本的例子程序。

程序清单 9-2　　使用用户字体绘制文本

```
#include <qapplication.h>
#include <qwidget.h>
#include <qpainter.h>
#include <qfont.h>

class MyMainWindow : public QWidget
{
public:
        MyMainWindow();
private:
        void paintEvent( QPaintEvent* );
        QPainter *paint;
};

void MyMainWindow::paintEvent( QPaintEvent* )
{
        paint = new QPainter;
        paint->begin( this );
        //Set the font, size and style:
        paint->setFont( QFont( "Arial", 16, QFont::Bold ) );
        //Draw the text:
        paint->drawText( 20, 20, 260, 60, AlignCenter,
```

```
23:                                "Font: Arial, Size: 16, Style: Bold" );
24:         paint->end();
25: }
26:
27: MyMainWindow::MyMainWindow()
28: {
29:         setGeometry( 100, 100, 300, 100 );
30: }
31:
32: void main( int argc, char **argv )
33: {
34:         QApplication a(argc, argv);
35:         MyMainWindow w;
36:         a.setMainWidget( &w );
37:         w.show();
38:         a.exec();
39: }
```

这里，你看到了第 15 行到第 25 行间对 paintEvent()函数的定义。这个定义中首先创建 QPainter 对象（第 17 行）。之后，在第 18 行开始绘图。第 19 行调用 QPainter::setFont()函数设置字体和样式。第 22 和 23 行调用 QPainter::drawText()函数绘制文本。传递给该函数 6 个参数。前两个参数定义绘制文本矩形的左上角。接下来的两个参数表示矩形的宽度和高度。第 5 个参数设置文本的对齐方式。最后一个参数说明所绘制的文本。这个程序创建一个窗口，之后，在其中心位置绘制下面文本：

```
Font: Arial, Size: 16, Style: Bold
```

图 9-5 为该程序执行结果。

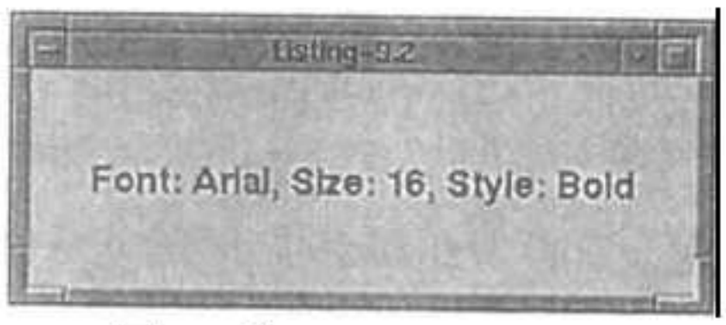


图 9-5　使用 QPainter 绘制文本



使用标签也能够很容易地达到这一目的。但是，这一课是介绍图形，你需要了解 QPainter 所提供的文本函数。



QPainter 具有与 QLabel 相同的对齐选项。

9.1.3　QPainter 绘图函数

QPainter 提供很多文本和矩形绘制函数。这一节介绍其他一些有趣的绘图函数（但不是所有函数）。如果需要了解所有 QPainter 成员函数，请参看 Qt Reference Document 中的 QPainter 部分。



记住，下面所介绍的例子代码总是插入到 paintEvent()函数中的 begin()和 end()调用之间。

1．绘制圆和椭圆

为了使用 QPainter 绘制圆和椭圆，应该使用 drawEllipse()函数。它带 4 个整数型参数。第一、二个参数分别表示圆/椭圆距窗口左上角的像素数。第 3、4 个参数表示圆/椭圆的宽度和高度。请看下面例子：

```
paint->setPen( blue, 4, QPen::SolidLine ) ;
paint->drawEllipse( 20, 20, 210, 160 ) ;
```

该代码的显示结果如图 9-6 所示。

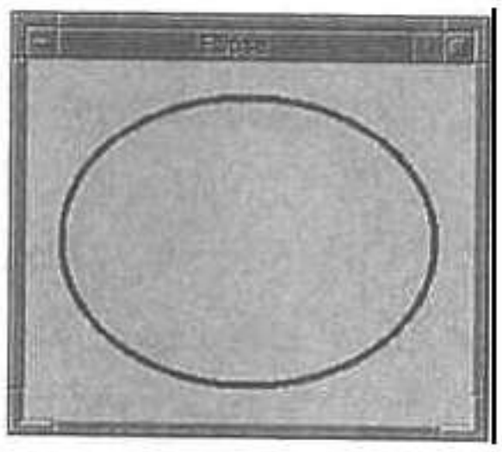


图 9-6　用纯色线绘制椭圆

2．绘制圆角矩形

为了绘制圆角矩形，需要使用 QPainter::drawRoundRect()函数。请看下面一段代码和图 9-7 中的显示结果：

```
paint->setPen( QPen(red, 4, QPen::SolidLine ) ) ;
paint->drawRoundRect ( 20, 20, 210, 160, 50, 50 ) ;
```

最后两个参数决定角的圆度。它可为 0 到 99 之间的任意值（99 代表最圆）。

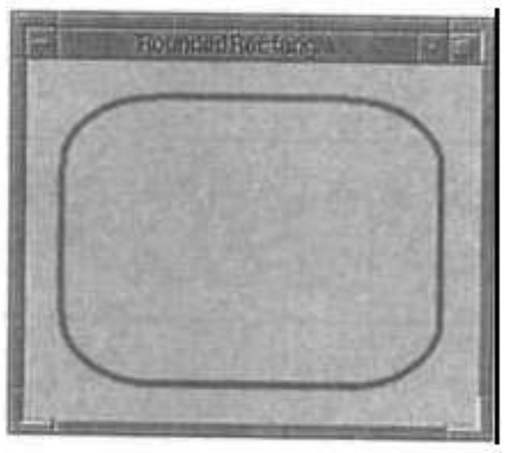


图 9-7　圆角矩形

3．绘制扇形图

为了绘制扇形图，应该使用 QPainter::drawPie()函数。这里是一个例子：

```
paint->setPen( QPen(green, 4, QPen::SolidLine ) ) ;
paint->drawPie ( 20, 20, 210, 160, 0, 500 ) ;
```

前 4 个参数定义圆（与 drawEllipse()函数相同）。后两个参数定义圆的样式。0 为起始角度（实际单位为 1/16 度），500（单位也为 1/16 度）为扇形图的弧线长度。如果你不熟悉三角几何，可能难以理解这些。如果是这样的话，应该用几个不同值测试一下。你很快就能

够理解其工作方式。图 9-8 为该程序执行结果。

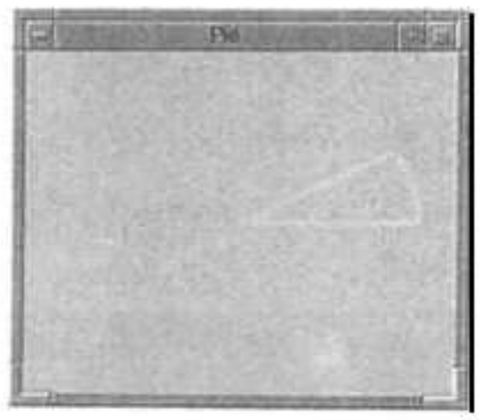


图 9-8　一个绿色扇形图



最后两个参数都不表示度，而是代表 1°的$\frac{1}{16}$。因此，在这种情况下，一个圆就表示为 5760（16 × 360）。

4. 绘制弦

为了得到一个弦，需要在圆里绘制一条直线，这条线就是弦。但是，QPainter 将绘制这条线外的一段圆弧。绘制弦需要使用 QPainter::drawChord()函数：

```
paint->setPen( QPen(green, 4, QPen::SolidLine ) ) ;
paint->drawChord( 20, 20, 210, 160, 500, 1000 ) ;
```

drawChord()函数的参数与 drawPie()函数的参数完全相同。图 9-9 为该代码执行结果。

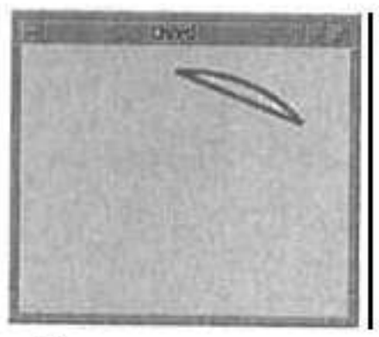


图 9-9　一个绿色弦

5. 绘制圆弧

用 QPainter 很容易绘制圆弧——只需要使用 QPainter::drawArc()函数。该函数的参数与 drawPie()和 drawChord()完全相同。例如：

```
paint->setPen( QPen(green, 4, QPen::SolidLine ) ) ;
paint->drawArc( 20, 20, 210, 160, 500, 1000 ) ;
```

图 9-10 为该代码的执行结果。

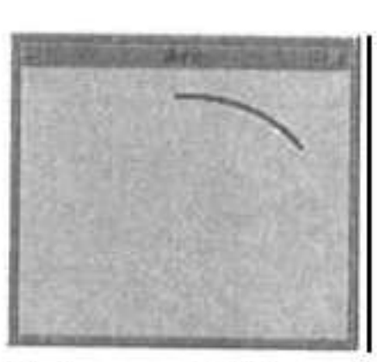


图 9-10　一个绿色圆弧

6. 绘制贝塞尔曲线

为了绘制贝塞尔曲线（由 4 个点描述的一条曲线），应该使用 QPainter::drawQuadBezier()

函数。下面是一个简单例子：

```
paint->setPen( QPen(green, 4, QPen::SolidLine ) ) ;
paint->drawQuadBezier ( QPointArray( QRect( 20, 20, 210, 160 ) ) ) ;
```

传递给该函数的唯一一个参数表示一个矩形，在其中创建贝塞尔曲线（其他参数为预定义参数，可以省略）。图 9-11 显示出一条贝塞尔曲线例子。

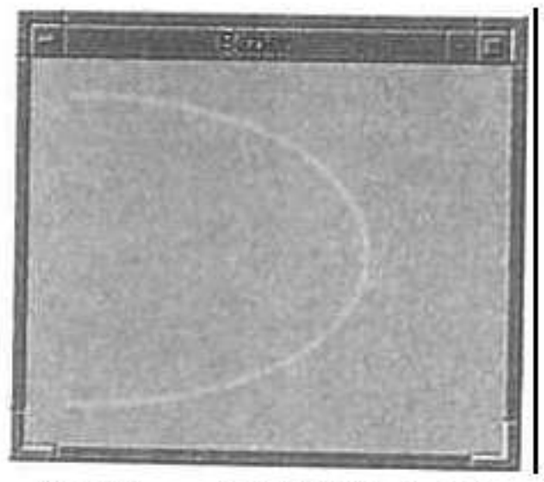


图 9-11　一条绿色贝塞尔曲线

你已经学习了几个 QPainter 函数。但是，还有很多这样的函数，如 QPainter::drawPixmap()（绘制位图）和 QPainter::drawPolygon()（绘制多边形）等。关于这些函数的更多信息请查阅 Qt Reference Document。

9.2　使用颜色

为了使应用程序看起来更美观，应该了解怎样管理颜色。由于硬件种类很多，管理颜色可能有一点棘手。但是，Qt 使这项工作变得更加简单。

9.2.1　管理颜色

当为应用程序选择颜色时，要始终记住应用程序的许多用户一次能够访问的颜色种类可能不超过 256 种（8bit 颜色深度）。因此，不要分配太多用户颜色，而要尽量使用那些其他应用程序已经分配的颜色。这样做，能够留出更多的色元（每种颜色占一个色元）。

当然，不可能知道哪种颜色已经使用，哪种颜色还没有使用。但是，应该习惯于使用 Qt 的预定义颜色——black（黑）、white（白）、red（红）、green（绿）、blue（蓝）、cyan（青）、yellow（黄）、magenta（紫）、gray（灰）、darkGrey（深灰）、lighGray（浅灰）、darkRed（深红）、darkGreen（深绿）、darkBlue（深蓝）、darkCyan（深青）、darkMagenta（深紫）、darkYellow（深黄）、color0 和 color1。这些颜色很可能已经使用。



color0 和 color1 是两种特殊的颜色，它们用于绘制双色位图。并能很好保证二者间的反差。

避免使用太多颜色种类的另外一种方法是使用颜色模式（给定几种颜色的集合）。使用 QApplication::setColorSpec()函数可以设置颜色模式。下面是几个传递给该函数的标准参数：

- QApplication::Normal：这是默认取值，如果应用程序主要使用标准颜色，则应该使

用该值：

● QApplication::ManyColors：如果应用程序需要使用很多种颜色，则使用该值。它使用一种特殊的方法（它使你的应用程序中能够为其他应用程序留出更多颜色）判断实际上应该分配哪一种颜色。该选项只能用于 UNIX 系统；

● QApplication::CustomColors：这只能应用于 Microsoft Windows 系统。它将更多的颜色分配给当前活动程序，而将较少颜色分配给后台应用程序。如果将 Windows 设置为 256 种颜色，你很可能已经看到不活动应用程序的颜色变得非常奇怪。这就是由这种方法引起的。



必须在创建 QApplication 对象之前调用 QApplication::setColorSpec()函数！

9.2.2　指定颜色

在 Qt 中，可以使用 3 种方式指定颜色：RGB（红/绿/蓝）模式、HSV（色度/饱和度/纯度）模式或者命名颜色方式。

一、RGB 模式

RGB 是最常用的颜色定义方法，你很可能在此之前已经听说过它。RGB 使用 3 个整数值定义颜色——3 个值分别代表红、绿、蓝。3 种颜色中哪一种值越大，它在这种颜色显示结果中所占的比重就越大。每种颜色值必须是 0 到 255 之间的整数。例如，（0, 0, 0）表示黑色，而（255, 255, 255）则代表白色。

为了在 Qt 中使用 RGB 分配一种颜色，需要创建一个 QColor 对象，并将 3 个 RGB 值作为参数传递给其构造函数。例如：

```
QColor  myBlack( 0, 0, 0) ;
```

这使 QColor 对象为黑色。如果想要一个纯红色。则用下面语句：

```
QColor  myRed( 255, 0, 0) ;
```

二、HSV 模式

HSV（Hue Saturation Value）模式是另外一种颜色定义方法。HSV 将一种颜色分为色度、饱和度和纯度。但是，所指定的数值有一点不同。在 Qt 中，色度可为-1 到 360 之间的任意整数，色饱和度和纯度可为 0 到 255 之间的任意整数。

在 QColor 中也可以使用 HSV，只需将 QColor::Hsv 添加到 QColor 构造函数中即可。例如：

```
QColor  hsvColor( 150, 73, 213, QColor::Hsv ) ;
```

三、命名颜色方式

所有的 X11（UNIX）系统均有一个将颜色名称映射为 RGB 值的数据库。使用这些命名颜色，使选择已使用颜色的机会变得更大，从而可以节省一些色元。在 QColor 中也可使用命名颜色。例如：

```
QColor  lBlue( "SteelBlue" ) ;
```



Qt 也提供一个用于 Windows 系统的颜色数据库。

四、修改标准调色板

如果不喜欢标准调色板，你可以定义一个用户调色板。一个调色板包含 3 个颜色组：活动组（当应用程序获得焦点时使用）、停用组（当禁止用户使用时使用）和正常组（用于所有其他目的）。每个颜色组中均由大量能够用于应用程序某一部分的颜色组成。

因此，如果想创建自己的调色板，则需要创建 3 个颜色组。这一功能由 QcolorGroup 类实现，将颜色作为参数传递给 QcolorGroup 构造函数（有关这些参数的描述请参看 Qt Reference Document。）。接下来，需要创建 Qpallete 对象，并将 3 个颜色组作为参数传递给 Qpallete 构造函数。这方面更详细信息请再参看 Qt Reference Document。

9.3　用 Qt 打印图形

在 Qt 中打印图形非常简单。Qprinter 类用于打印，因为它是 QpaintDevice 类（与 Qwidget 一样）的子类，所以，能够在 Qprinter 对象上使用 Qpainter 绘图。之后，所绘制的所有内容都被打印出来。唯一需要控制的事情是什么时候换页。这由 Qprinter::newPage()函数实现。

然而，你可能希望让用户能够修改打印机设置。调用 Qprinter::getPrinter()函数即可显示出打印机设置对话框。这个对话框如图 9-12 所示。

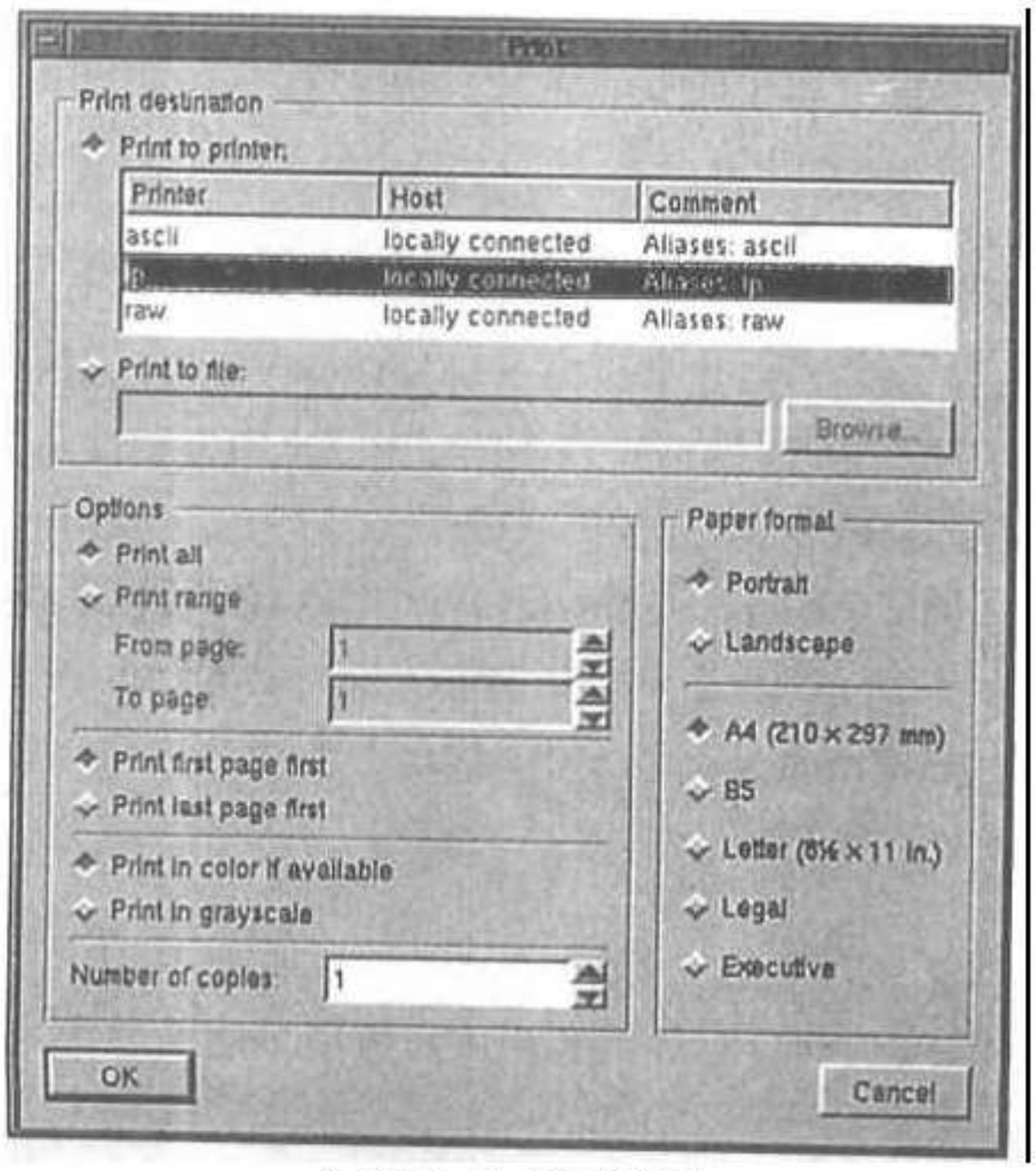


图 9-12　Qt 打印对话框



如果你使用 Windows 平台，那么打印机设置对话框将随你所使用的打印机不同而有所改变。

如果点击 OK 按钮，Qprinter::setup()函数将返回 true。因此，如果函数返回 true，则应继续打印。如果返回值不为 TRUE，打印将被取消。下面例子说明这一操作：

```
Qprinter print ;
int proceed = print_setup( );
if ( proceed == true )
{
    //start printing
    //(e.g. Start drawing to the
    //Qprinter object)
}
else
{
    //Abort
}
```

它是如此简单！当然，也可以通过编程来设置不同的打印选项。使用 Qprinter::setOrientation()函数（向它传递 Qprinter::Portrait 或 Qprinter::Landscape）能够设置打印方向。也可调用 Qprinter::setMinMax()函数设置打印的最多和最少页数。

9.4　小　结

Qt 带有很多绘图函数，这一课中并没有全部介绍。剩余部分将在第 15 学时“深入学习图形”中介绍。

在这一课最后一节中，学习了怎样在 Qt 应用程序中实现图形打印功能。尽管 Qt 提供几个重要的打印功能，但是，应该记住这一工作与平台有很大关系。因此，需要有合适的驱动程序来配置系统，否则，将无法打印。

9.5　问题与答案

问：当我向类构造函数添加一些绘图代码后，屏幕上未显示出我所绘制的图形。这是为什么？

答：如前所述，应该将绘图代码放置在一个称做 paintEvent()的特殊函数内。这能够避免用户代码干扰 Qt 内部绘图代码。但是，也可以将绘图代码放置在一些其他函数内，但不要在构造函数中。

问：当我使用 Qpainter 绘制文本时，显示出未找到字体错误。这是为什么？

答：实际上，你已经找到了错误原因。系统没有安装你所选择使用的字体类型，或是拼写错误。

问：为什么在我试图打印图形时未打印出任何内容？

答：确保打印机已正确安装。这是一个很常见的问题。再检查向打印机输出图形的代码中是否存在错误。最后，要确保已经调用 begin()函数，并使用 Qpainter 对象作为它的一个参数。

9.6 作　业

完成下面问题和练习将有助于牢记本课所学内容。通过完成这些问题和练习，也能够使你确实理解在程序中怎样处理图形。

9.6.1 测验

1. 什么类用于创建图形？
2. setPen()函数的功能是什么？
3. 画刷有什么用途？
4. 是否有多种填充模式供选择使用？
5. 为什么要使用 QPainter 类显示文本？
6. 什么是 RGB？
7. 什么是调色板？
8. 在纸上打印出一个圆需要做哪些工作？

9.6.2 练习

1. 创建一个绘制圆（而不是椭圆）的应用程序。其外框线为红色、虚线和 5 个像素宽。使用纯蓝色填充该圆。
2. 假设今天是你的生日，妈妈为你制作了一个大蛋糕，并且切下其中的一大块。绘制所切下的这块蛋糕。
3. 编写一个具有打印功能的应用程序。创建一个标签为“Print”的按钮。当点击这个按钮时，显示出打印对话框。之后，当点击 OK 按钮时，程序应该打印出你在前一个程序中所创建的图形，并在其下面添加“This slice comes from a really nice cake!”文本。

第 10 学时　理解 Qt 对话框

如果想以某种方式与用户交互——例如，想让他们选择一个在文本编辑器中打开的文件——则需要使用对话框来实现。

Qt 提供很多用于常用对话框，例如，选择文件等。在很多情况下，这些预定义对话框能够实现一些功能，使用它们可以节省大量的工作量。事实上，在很多情况下，只需要一行代码就足以实现一个预定义对话框。

当然，如果需要一个很特殊的对话框，则必须自己进行创建。但是，使用 Qt 提供的对话框创建工具，这一工作对你来说也不是太难。

在这一课里，将学习怎样使用预定义对话框和怎样创建自己的对话框。

10.1　预定义对话框

Qt 提供几个对话框，它们是 QColorDialog、QFileDialog、QFontDialog、QMessageDialog 和 QProgressDialog。这些类能够用于创建最常用的对话框，如选择文件、字体或颜色，或向用户提问 Yes/No 之类的问题和通知他们程序所发生的事情。最后一个对话框类，QProgressDialog，能够用于创建进度对话框，它向用户通知程序的当前状态。这些对话框都是最常用的对话框，因此，它们被预定义在 Qt 库中。这一节逐个介绍这些对话框。

10.1.1　颜色对话框

QColorDialog 提供一个用于选择颜色的对话框。可调用该类的唯一一个成员函数 getColor()来显示对话框。这个函数返回所选中的颜色（实际上为一个表示颜色的 QColor 对象）。程序清单 10-1 给出一个例子：

程序清单 10-1　　颜色对话框

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qcolordialog.h>
4: #include <qcolor.h>
5:
6: class MyMainWindow : public QWidget
7: {
8: public:
```

```
 9:          MyMainWindow();
10: private:
11:          QColorDialog *cdialog;
12:          QColor myColor;
13: };
14:
15: MyMainWindow::MyMainWindow()
16: {
17:          setGeometry( 100, 100, 200, 50 );
18:
19:          //Select a color and store it
20:          //in myColor for further use:
21:          myColor = cdialog->getColor( QColor( 0, 0, 0 ) );
22: }
23:
24: void main( int argc, char **argv )
25: {
26:          QApplication a(argc, argv);
27:          MyMainWindow w;
28:          a.setMainWidget( &w );
29:          w.show();
30:          a.exec();
31: }
```

这里，QColorDialog::getColor()函数的返回值被存储到 myColor 种。这样做，你能够很容易地使用这个颜色。例如，可以将这个颜色设置为绘图程序中的颜色。传递给 getColor()函数的颜色为预选颜色。图 10-1 显示的就是颜色对话框。

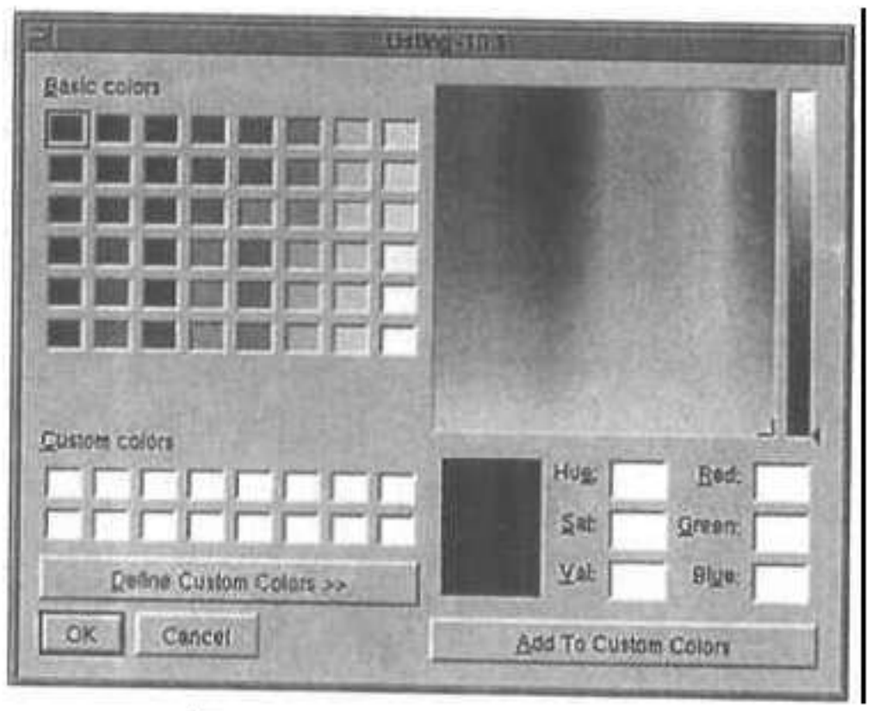


图 10-1 Qt 提供的颜色对话框

但是，这里还需要执行另外一个任务。当用户点击颜色对话框中的 Cancel 按钮时，getColor()函数将返回一个无效颜色，myColor 就不代表实际颜色。但是，程序中能够很容易地检查出这种情况。例如：

```
if ( myColor.isValid( ) == TRUE )
{
    //myColor is okay and ready
    //to be used.
}
```

```
if ( myColor.isValid( ) == FALSE )
{
    //myColor is not valid and
    //we must therefore abort.
}
```

正如你所看到的，使用 QColor::isValid()函数判断 myColor 是否有效。第 1 个 if 语句检查所选择的颜色是否有效，如果它为 true，应用程序能够继续并使用这个颜色。第 2 个 if 语句（它应当简化为 else 语句，但是为了使程序更加清晰使用了 if 语句）用于检查所选择的颜色是否为无效颜色。如果是这样，程序将放弃操作。这就避免出现问题。如果选择一种无效颜色，程序不能再使用它，在最坏情况下，这可能导致程序崩溃。

10.1.2　文件对话框

QFileDialog 提供一个用于选择文件的对话框。与 QColorDialog 对话框一样，它也非常容易使用，并可由单行代码实现。程序清单 10-2 就是一个例子：

程序清单 10-2　　文件对话框

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qfiledialog.h>
4: #include <qstring.h>
5:
6: class MyMainWindow : public QWidget
7: {
8: public:
9:         MyMainWindow();
10: private:
11:         QFileDialog *fdialog;
12:         QString file;
13: };
14:
15: MyMainWindow::MyMainWindow()
16: {
17:         setGeometry( 100, 100, 200, 50 );
18:         //Select a file and store it in
19:         //file for further use:
20:         file = fdialog->getOpenFileName( "/", "*.txt" );
21: }
22:
23: void main( int argc, char **argv )
24: {
25:         QApplication a(argc, argv);
26:         MyMainWindow w;
27:         a.setMainWidget( &w );
28:         w.show();
29:         a.exec();
30: }
```

这里，第 11 行为 QFileDialog 对象分配内存。之后，第 20 行（对话框在这里被显示到屏幕上）执行对话框。在文件对话框中所选中的文件名（包括全部路径）被存储到一个 QString

对象 file（在第 12 行创建）中。需要向 QFileDialog::getOpenFileName()函数传递两个参数（第 20 行）。第 1 个参数表示文件系统的起点（Windows 用户可以将它修改为 c:\之类的字符串），第 2 个参数为所使用的过滤器。在这里，你只能看到以.txt（文本文件）结尾的文件。图 10-2 显示出文件对话框。

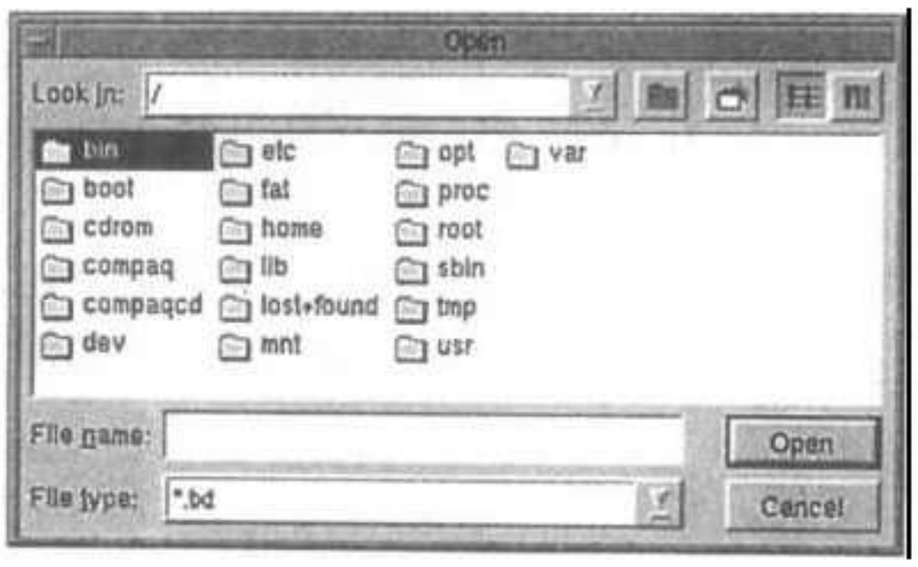


图 10-2　Qt 提供的文件对话框

此外，用户可以点击 Cancel 按钮来取消文件选择操作（与颜色对话框一样）。因此，需要用一些代码来处理这种情况：

```
if ( file.isNull( ) == FALSE )
{
    //A file name was found and
    //we can processed
}
if ( file.isNull( ) == TRUE )
{
    //A file name is NULL
    //and we must abort.
}
```

这里，QString.isNull()函数判断是否选中一个文件。

QfileDialog 还包含一些其他函数：QFileDialog::getExistingDirectory()使用户选择一个目录，QFileDialog::getOpenFileName()函数使用户能够选择多个文件。有关这些函数的更多信息请参看 Qt Reference Document。

10.1.3　字体对话框

在 Qt 中，QFontDialog 类用于实现字体对话框。使用 QFontDialog::getFont()函数显示该对话框。之后，这个函数返回一个 QFont 对象，它表示新选择的字体。程序清单 10-3 给出一个例子：

程序清单 10-3　　字体对话框

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qfontdialog.h>
4: #include <qfont.h>
5:
```

```
 6: class MyMainWindow : public QWidget
 7: {
 8: public:
 9:         MyMainWindow();
10: private:
11:         QFontDialog *fdialog;
12:         QFont myFont;
13: };
14:
15: MyMainWindow::MyMainWindow()
16: {
17:         setGeometry( 100, 100, 200, 50 );
18:
19:         //ok will be TRUE if the user
20:         //selected a font:
21:         bool ok;
22:
23:         //Select a file and store it in
24:         //file for further use:
25:         myFont = fdialog->getFont( &ok );
26: }
27:
28: void main( int argc, char **argv )
29: {
30:         QApplication a(argc, argv);
31:         MyMainWindow w;
32:         a.setMainWidget( &w );
33:         w.show();
34:         a.exec();
35: }
```

第 11 行和第 12 行分别为一个 QFontDialog 对象和一个 QFont 对象分配内存。第 21 行创建一个布尔变量。然后在第 25 行使用该变量，并在这里调用 QFontDialog::getFont()函数显示字体对话框。在第 25 行，被选中的字体（QFontDialog::getFont()函数的返回值）被存储到 QFont 对象 myFont 中。字体对话框如图 10-3 所示。

为了判断用户是否已经选择一种字体，你需要创建一个布尔变量（像程序清单 10-3 中那样），并将该变量的内存地址传递给 QFontDialog::getFont()函数。如果用户选择一种新的字体，这个变量（这里为 ok）的值将为 TRUE，如果用户点击 Cancel 按钮，该变量的值将为 FALSE。下面一段代码说明怎样处理这种情况：

```
if ( ok == TRUE )
{
    //A new font was selected,
    //and we can proceed.
}
if ( ok == FALSE )
{
    //the user clicked the cancel
    //button, we better abort
}
```

如果用户选择一种新的字体，可使该字体存储到 myFont 中，之后就可以任意地使用它。例如，用于定义 QLabel 对象中的字体等。

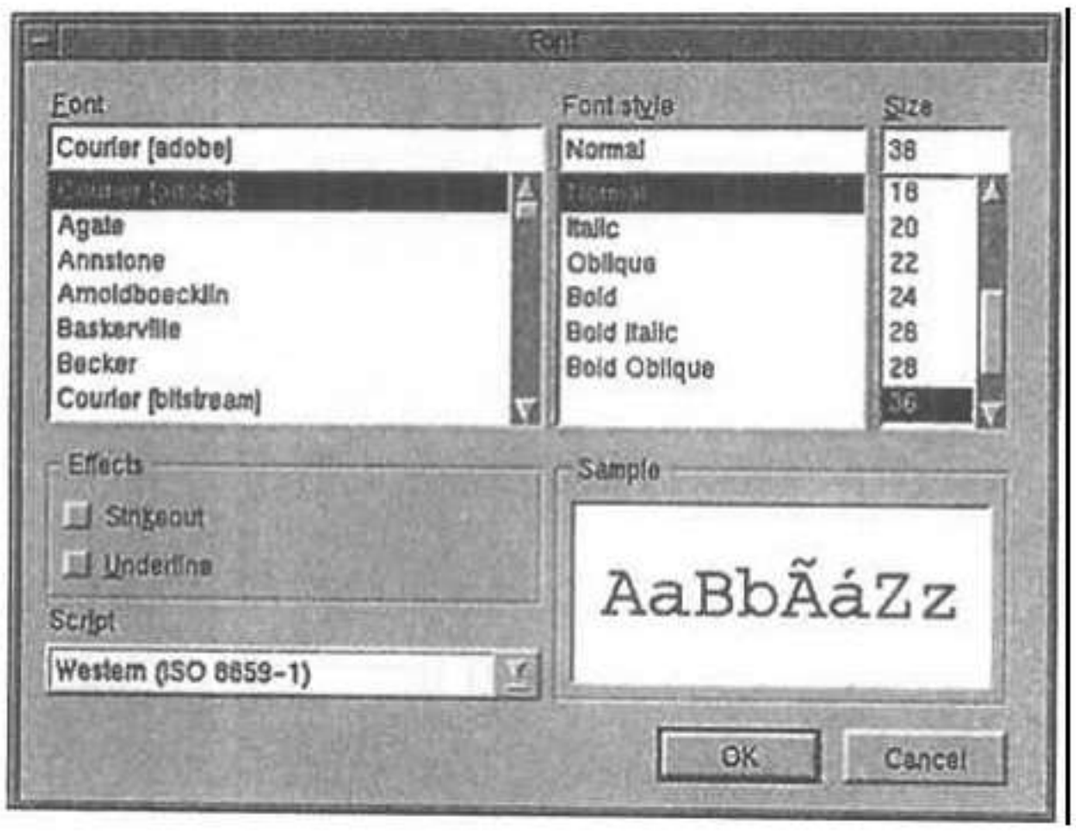


图 10-3　Qt 提供的字体对话框

10.1.4　消息对话框

QMessageDialog 类提供的消息对话框是最简单的 Qt 预定义对话框。它可用于显示一些短信息（像在前一课中演示的那样），以提问用户是否需要继续。消息框能够显示一个图标和多达 3 个按钮。程序清单 10-4 给出一个例子：

程序清单 10-4　　消息对话框

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qmessagebox.h>
 4:
 5: class MyMainWindow : public QWidget
 6: {
 7: public:
 8:         MyMainWindow();
 9: private:
10:         QMessageBox *mbox;
11: };
12:
13: MyMainWindow::MyMainWindow()
14: {
15:         setGeometry( 100, 100, 200, 50 );
16:         mbox = new QMessageBox( "Proceed?", "Do you want to proceed?",
17:                                 QMessageBox::Critical,
18:                                 QMessageBox::Ok | QMessageBox::Default,
19:                                 QMessageBox::Cancel | QMessageBox::Escape,
20:                                 0 );
21:         mbox->show();
22: }
23:
24: void main( int argc, char **argv )
25: {
```

```
26:         QApplication a(argc, argv);
27:         MyMainWindow w;
28:         a.setMainWidget( &w );
29:         w.show();
30:         a.exec();
31: }
```

这里，使用 QMessageDialog 构造函数创建消息框（第 16 行到 20 行）。第 21 行调用 QMessageDialog::show()函数，它在屏幕上显示出消息框。与你所看到的一样，QMessageDialog 构造函数需要好几个参数。第 1 个参数（第 16 行）表示需要在消息框上边显示的文本字符串（也称做窗口标题）。第 2 个参数（第 16 行）为在消息框内显示的文本字符串（实际消息）。第 3 个参数（第 17 行）为在消息框内所显示的图标。这里，有 4 种选择：

- QMessageBox::NoIcon：不显示任何图标；
- QMessageBox::Information：显示一个信息图标；
- QMessageBox::Warning：显示一个警告信息图标；
- QMessageBox::Critical：如果要显示一些关键信息，或是希望用户做出关键选择时，使用该图标。

这里使用 QMessageBox::Critical 图标（第 17 行）。第 4 个参数（第 18 行）定义第 1 个按钮，这里使用 QMessageBox::OK 按钮。它与 QMessageBox::Default 结合使它变为默认按钮（当用户按回车键时将会“点击”它）。第 5 个参数（第 19 行）定义第 2 个按钮。这里选择 Cancel 按钮。这个按钮与 QMessageBox::Escape 结合。这使得用户在按下 Escape 键时将会“点击”该按钮。



完全可以不将按钮与 QMessageBox::Default 或 QMessageBox::Escape 相结合。

最后一个参数定义第 3 个按钮。但是，这个例子中只使用两个按钮，因此，这里输入 0。该代码得运行结果如图 10-4 所示。

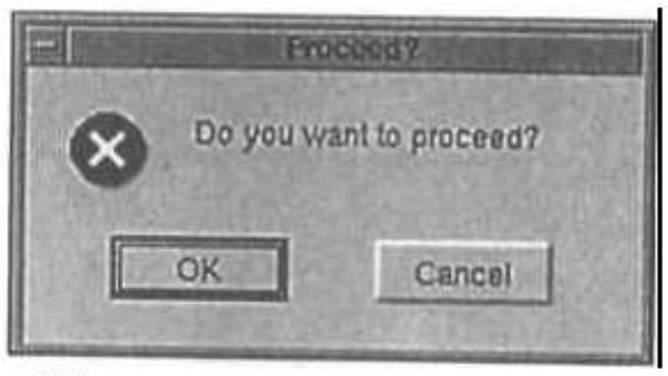


图 10-4　具有两个按钮、一条短信息和一个图标的消息框

当然，除了 OK 和 Cancel 按钮外，还可选择其他许多。这些按钮包括：QMessageBox::Yes、QMessageBox::No、QMessageBox::Abort、QMessageBox::Retry 和 QMessageBox::Ignore。

10.1.5　进度对话框

进度对话框用于将程序的当前状态（与进度条一样）通知用户。进度对话框与进度条间的区别是进度对话框能够与用户进行交互（也就是说，用户能够中断他或她想中断的任务）。来看程序清单 10-5 所给出的例子：

程序清单 10-5　　进度对话框

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qprogressdialog.h>
 4:
 5: class MyMainWindow : public QWidget
 6: {
 7: public:
 8:     MyMainWindow();
 9: private:
10:     QProgressDialog *pdialog;
11: };
12:
13: MyMainWindow::MyMainWindow()
14: {
15:     setGeometry( 100, 100, 200, 50 );
16:
17:     pdialog = new QProgressDialog( "Doing something...",
18:                         "Abort Operation", 100,
19:                             this, "pdialog", TRUE );
20:     pdialog->show();
21:
22:     int x = 0;
23:     int i = 0;
24:     while( x <= 100 )
25:     {
26:         for( i = 0; i < 1000000; i++ );
27:         pdialog->setProgress(x);
28:         x++;
29:     }
30: }
31:
32: void main( int argc, char **argv )
33: {
34:     QApplication a(argc, argv);
35:     MyMainWindow w;
36:     a.setMainWidget( &w );
37:     w.show();
38:     a.exec();
39: }
```

总共有 6 个参数传递给 QProgressDialog 构造函数（第 17、18 和 19 行）。第 1 个参数（第 17 行）表示在进度对话框上端显示的文本（不是标题）。它将程序正在执行的操作告诉用户。第 2 个参数（第 18 行）表示在“取消”按钮上显示的文本。第 3 个参数（第 18 行）为总的步进数量。在这里，当调用 QProgressBox::setProgress(100)时进度条将达到 100%。最后 3 个参数具有预定义值，它们可以省略。但是，如果希望当进度对话框开始工作时能够重新绘制，则需要将第 6 个参数设置为 TRUE。这时，也需要定义第 4 和第 5 个参数。如果不需要将第 6 个参数设置为 TRUE，将看不到取消按钮上的标签。

为了演示怎样使用进度对话框，我建立了一个循环。它每次使用一个更大的值调用 setProgress()函数。在实际程序中你肯定不需要这样做。但这只是为了显示其工作方式（见图 10-5）。



图 10-5　这里的进度条是活动的，其当前值为 57%

10.2　创建用户对话框

当预定义对话框不能满足需要时，就需要创建用户对话框——被特殊设计、能够满足你需要的对话框。Qt 提供两个类，它们使用户对话框的创建变得更加简单：QDialog（所有对话框的基类）和 QTabDialog（用于创建选项卡对话框）。

10.2.1　用 QDialog 创建用户对话框

QDialog 是基本的对话框类——所有其他对话框类都基于它。它提供几个的基本函数用于创建对话框。

当创建对话框时，有两种对话框类型供你选择：模式对话框和非模式对话框。模式对话框启动它自己的事件循环，这导致在对话框关闭之前它会阻塞程序的其余事件。相反，非模式对话框不会阻塞应用程序中的其他事件。用一个布尔值作为 QDialog 构造函数的第 3 个参数就能够控制所创建的对话框是模式对话框还是非模式对话框。TRUE 意味着模式对话框，FALSE 意味着非模式对话框。

现在，我们来看 QDialog 的几个重要函数。QDialog::setCaption()函数用于设置对话框窗口标题。QDialog::accept()、QDialog::reject()和 QDialog::done()函数均能够关闭对话框，并返回一个值。Accept()返回常量 QDialog::Accept(1)。Reject()返回常量 QDialog::Reject(0)。也可以向 done()函数传递一个值(整数)作参数来定义自己的返回值。事实上，可以调用 done(1)和 done(0)来代替 accept()和 reject()，其结果完全相同。

QDialog 另一个有趣的功能是不必使用 setGeometry()或 resize()来定义窗口大小。如果只是把部件放置在对话框里，QDialog 会自动调整窗口大小来适应这些子部件。程序清单 10-6 给出一个简单的例子，它说明怎样使用 QDialog 创建用户对话框。

程序清单 10-6　　用 QDialog 创建用户对话框

```
 1: #include <qapplication.h>
 2: #include <qdialog.h>
 3: #include <qlabel.h>
 4: #include <qpushbutton.h>
 5:
 6: class Question : public QDialog
 7: {
 8: public:
 9:     Question();
10: private:
11:     QLabel *label;
12:     QPushButton *big;
```

```
13:     QPushButton *small;
14: };
15:
16: //Here we define the dialog window. We pass three
17: //arguments to the QDialog constructor. The first
18: //two (parent and name) are set to 0. The third
19: //makes this dialog modal.
20: Question::Question() : QDialog( 0, 0, TRUE )
21: {
22:     setCaption( "Big or small?" );
23:
24:     label = new QLabel( this );
25:     label->setText( "Do you want to see a big
26:             or a small window?" );
27:     label->setGeometry( 10, 10, 300, 100 );
28:     label->setFont( QFont( "Arial", 18, QFont::Bold ) );
29:     label->setAlignment( AlignCenter );
30:
31:     big = new QPushButton( "BIG!", this );
32:     big->setGeometry( 50, 120, 90, 30 );
33:     big->setFont( QFont( "Arial", 14, QFont::Bold ) );
34:
35:     small = new QPushButton( "small!", this );
36:     small->setGeometry( 180, 120, 90, 30 );
37:     small->setFont( QFont( "Arial", 14, QFont::Bold ) );
38:
39:     connect( big, SIGNAL( clicked() ), this, SLOT( accept() ) );
40:     connect( small, SIGNAL( clicked() ), this, SLOT( reject() ) );
41: }
42:
43: class MyProgram : public QWidget
44: {
45: public:
46:         MyProgram();
47: private:
48:         Question *q;
49: };
50:
51: MyProgram::MyProgram()
52: {
53:     q = new Question();
54:     //Because Question is a modal dialog,
55:     //we call exec() instead of show():
56:     int i = q->exec();
57:
58:     //If the user clicks the button
59:     //labeled "BIG!", show a big window:
60:     if( i == QDialog::Accepted )
61:     {
62:         resize( 500, 500 );
63:     }
64:
65:     //If the user clicks the button
66:     //labeled "small!", show a small window:
67:     if( i == QDialog::Rejected )
```

```
68:     {
69:         resize( 50, 50 );
70:     }
71: }
72:
73: void main( int argc, char **argv )
74: {
75:     QApplication a(argc, argv);
76:     MyProgram w;
77:     a.setMainWidget( &w );
78:     w.show();
79:     a.exec();
80: }
```

如你所看到的，这个程序由两个类组成：一个基于 QDialog（第 6 行到第 41 行），另一个基于 QWidget（第 43 行到第 71 行）。在 Question 构造函数中，它将对话框设置为拥有一个标签（第 24 行到第 29 行）和两个按钮（第 31 行到第 37 行）。在第 39 行和第 40 行，两个按钮的 clicked()信号分别被连接到 QDialog 的 accept()和 reject()槽。这确保对话框能够返回正确的常量（它在第 56 行被存储到 i 中）。基于 QWidget 的类（MyProgram）是主部件，从这里开始创建一个 Question 类对象（第 48、53 和 56 行）。之后，在第 60 行到第 70 行检查 Question 对象(q)的返回值，并相应设置窗口大小。如果你已经学习并理解了前面各课中的内容，那么在理解这个程序时就不会有任何问题。代码中的注释给出更进一步的解释。当运行这个程序时，首先看到的就是图 10-6 所示的对话框。



图 10-6　程序清单 10-6 中 Question 类所创建的对话框

现在，如果你点击“BIG!”按钮，将显示一个非常大的窗口。相反，如果你点击“small!”按钮，则将显示一个小窗口。这些窗口分别如图 10-7 和图 10-8 所示。

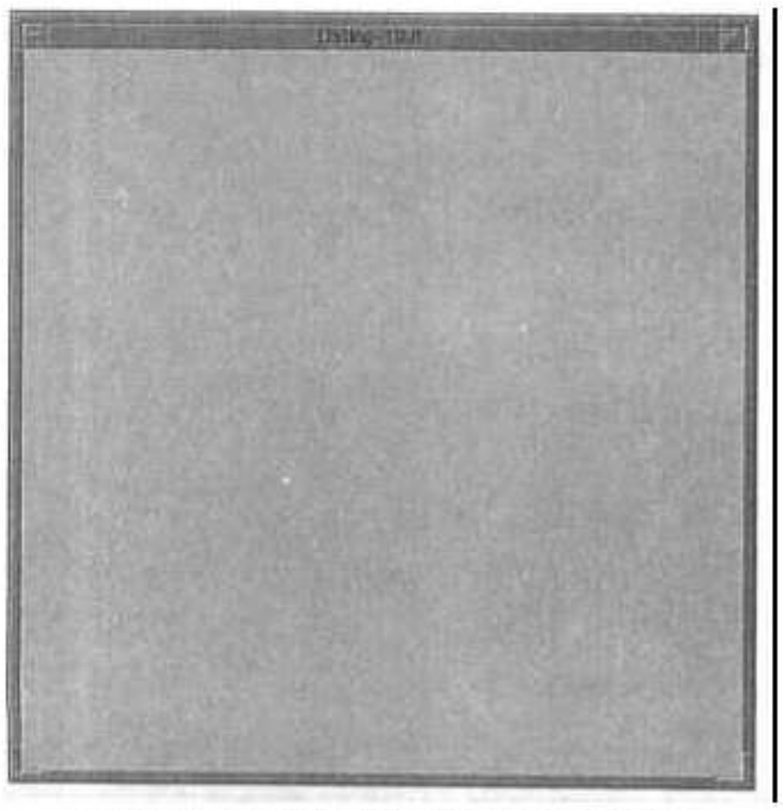


图 10-7　点击 BIG!按钮后所显示的窗口



图 10-8　点击 small!按钮后所显示的窗口

你现在应该对怎样使用 QDialog 有一个很好的理解。使用 QMessageBox 也能够创建程序清单 10-6 这个例子。但是，在知道怎样使用 QDialog 后，你将发现创建满足特殊需要的对话框非常容易。但是，还是应尽可能地使用预定义对话框，以节省你的时间和工作量。但当这些对话框不能完成你要做的工作时，则必须自己使用 QDialog 来创建它。

10.2.2　选项卡对话框

能够用于创建用户对话框的另一个 Qt 类是 QTabDialog。使用它可以创建所谓的选项卡对话框。一个选项卡对话框是由几个页面组成的部件。在很多应用程序中你可能已经看到过这种类型的对话框。当有很多部件，并且将所有这些部件放置在一个页面中会令人感到混乱时，就应该使用选项卡对话框。用选项卡对话框可以将这些部件分为不同的类，在选项卡对话框中为每一类创建一个页面。这使应用程序看起来更整洁，也更容易使用。

程序清单 10-7 显示一个选项卡对话框的原始状态（当选项卡对话框中未添加任何选项卡时）：

程序清单 10-7　　原始状态下的 QTabDialog 对象

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qtabdialog.h>
 4:
 5: class MyMainWindow : public QWidget
 6: {
 7: public:
 8:         MyMainWindow();
 9: private:
10:         QTabDialog *tdialog;
11: };
12:
13: MyMainWindow::MyMainWindow()
14: {
15:         setGeometry( 100, 100, 200, 50 );
16:
17:         tdialog = new QTabDialog();
18:         tdialog->resize( 300, 250 );
19:         tdialog->show();
20: }
21:
22: void main( int argc, char **argv )
23: {
24:         QApplication a(argc, argv);
25:         MyMainWindow w;
26:         a.setMainWidget( &w );
27:         w.show();
28:         a.exec();
29: }
```

图 10-9 显示出这个空的选项卡对话框。

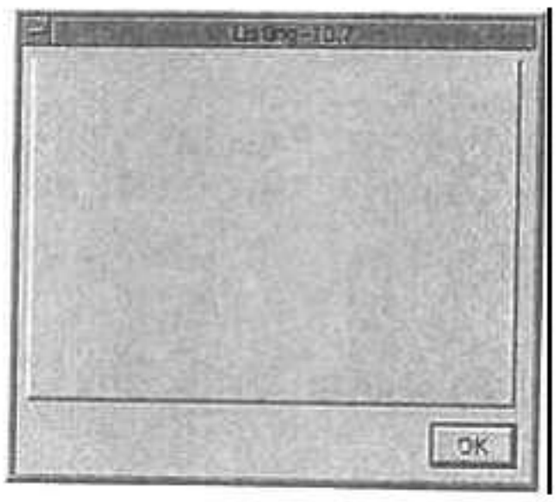


图 10-9　用 QTabDialog 创建的空选项卡对话框。点击 OK 按钮将关闭该选项卡对话框

如果想往选项卡对话框中添加部件，则必须在其上创建几个拥有子部件的 QWidget 对象，然后使用 QTabDialog::insertTab()函数将 QWidget 对象添加到选项卡对话框。程序清单 10-8 就是一个这样的例子：

程序清单 10-8　　拥有 3 个页面的选项卡对话框

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qtabdialog.h>
 4: #include <qlabel.h>
 5:
 6: class MyMainWindow : public QWidget
 7: {
 8: public:
 9:     MyMainWindow();
10: private:
11:     QTabDialog *tdialog;
12:     QWidget *page1;
13:     QWidget *page2;
14:     QWidget *page3;
15:     QLabel *label;
16: };
17:
18: MyMainWindow::MyMainWindow()
19: {
20:     setGeometry( 100, 100, 200, 50 );
21:
22:     tdialog = new QTabDialog();
23:     tdialog->resize( 300, 250 );
24:
25:     page1 = new QWidget();
26:     page1->resize( 280, 180 );
27:     label = new QLabel( "This is Page 1.", page1 );
28:     label->setGeometry( 20, 20, 240, 140 );
29:     label->setAlignment( AlignCenter );
30:
31:     page2 = new QWidget();
32:     page2->resize( 280, 180 );
33:     label = new QLabel( "This is Page 2.", page2 );
34:     label->setGeometry( 20, 20, 240, 140 );
35:     label->setAlignment( AlignCenter );
36:
37:     page3 = new QWidget();
38:     page3->resize( 280, 180 );
```

```
39:     label = new QLabel( "This is Page 3.", page3 );
40:     label->setGeometry( 20, 20, 240, 140 );
41:     label->setAlignment( AlignCenter );
42:
43:     tdialog->insertTab( page1, "Page 1" );
44:     tdialog->insertTab( page2, "Page 2" );
45:     tdialog->insertTab( page3, "Page 3" );
46:     tdialog->show();
47: }
48:
49: void main( int argc, char **argv )
50: {
51:     QApplication a(argc, argv);
52:     MyMainWindow w;
53:     a.setMainWidget( &w );
54:     w.show();
55:     a.exec();
56: }
```

这里，第 22 行和 23 行创建选项卡对话框。从第 25 行到第 41 行在选项卡对话框中创建 3 个各表示一个选项卡的 QWidget 对象。之后，在第 43、44 和 45 行使用 QTabDialog::insertTab()函数将 QWidget 对象添加到选项卡对话框中。在第 46 行，调用 QTabDialog::show()函数显示出选项卡对话框。图 10-10 为该程序执行结果。

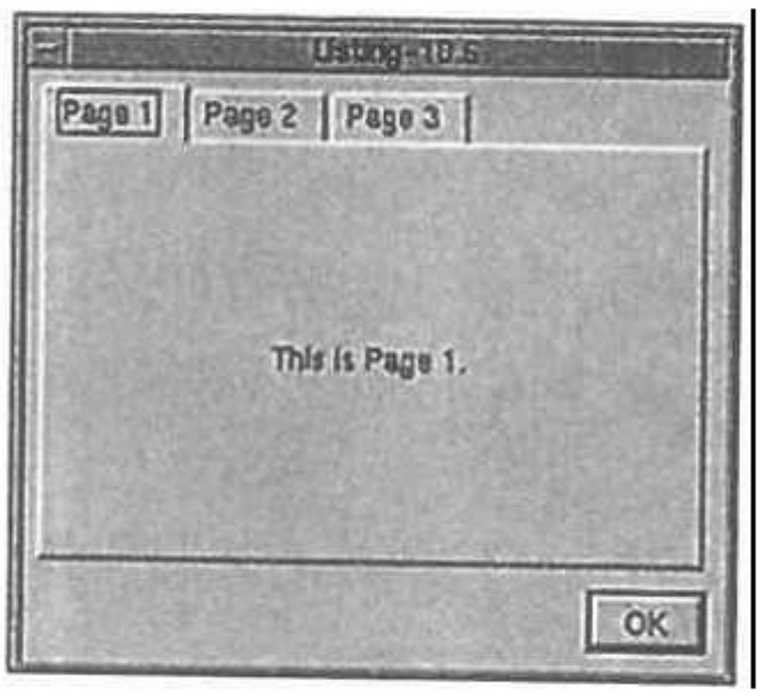


图 10-10　具有 3 个页面的选项卡对话框，每个页面上有一个标签

所有工作就这些。你只需像通常一样创建一个部件，之后再使用 QTabDialog::insertTab()函数插入它们。



当使用 insertTab()函数插入页面时，可以向标签的开始部分（或者在任何一个你要关联的快捷字符前面）添加一个&符号来自动创建一个快捷键。例如，我们来看下面一行：

```
tdialog->insertTab( page1, "&Options" );
```

在这种情况下，按键盘上的Ctrl+O 键你就能够访问Options 页面。这一功能也适用于其他很多Qt 部件（如菜单）。

10.3　小　结

在每一个重要的应用程序中，对话框都起着非常重要的作用。一个中等规模的应用程序可能需要创建 30 个以上的对话框，大型项目中或许会有上百个。

通常，Qt 预定义对话框就能完全满足你的需要。但是，当它们不能满足需要时，可以使用构造用户对话框的 Qt 类。更重要的是，它们能够为你做许多工作。

在这一课，已经学习了 Qt 预定义对话框和创建用户对话框。具有这些知识，你能够在 Qt 应用程序中创建所有类型的对话框。不管它们是常用对话框还是专用对话框。

10.4　问题与答案

问：在我的选项卡对话框底端，部件只有部分被显示出来。这是为什么？

答：如你所看到的，选项卡对话框未占用整个窗口，在它的四周有一些空间。如果你想使页面具有与窗口相同的尺寸，一些部件将无法显示出来，或者是只有部分可见。

问：我不能看到进度对话框中的标签或者是取消按钮。这是为什么？

答：需要将对话框设置为模式对话框。这通过将一个布尔值 TRUE 作为第 6 个参数传递给 QProgressBar 构造函数来实现。

问：消息对话框是不是必须具有 3 个按钮？我只想使用一、两个按钮！

答：不，不是必须具有 3 个按钮。只要用 0 代替第 3 个按钮的正常定义位置，并把它传递给 QMessageBox 构造函数即可。

10.5　作　业

这一节中关于本课的问题和练习将帮助你牢记已经学过的对话框知识。

10.5.1　测验

1. 什么是对话框？
2. Qt 是否提供文件选择对话框？
3. 是否能够选择多个文件？
4. 为什么你想让用户选择颜色？
5. 什么类用于创建字体对话框？
6. 如果你想向用户提问一个简单的问题，应该使用哪个对话框？
7. 如果你准备打开 50 个文件，你应该设置的最大进度值为多少？
8. 能使用 QDialog 基类创建自己的选项卡对话框吗？

10.5.2　练习

1. 使用字体对话框创建一个程序，使用户从中选择字体。之后，用用户所选择的字体显示一些文本。
2. 在练习 1 所创建的程序中实现一个颜色对话框。使用户选择颜色和字体，之后，使用 QPainter 类按照用户所选定的颜色和字体显示一些文本。
3. 创建一个对话框，提示用户输入其名字。用户应该能够在文本输入域中输入名字。之后，点击 OK 按钮（或类似按钮）提交。在此之后，程序响应用户输入，显示出一个具有以下信息的新窗口：Hello <name>, how are you today?
4. 创建一个选项卡对话框，向它添加一些页面，之后向这些页面中添加单选按钮、复选按钮和其他部件。如果其他程序中使用了选项卡对话框，试着使用 Qt 部件复制这些选项卡对话框（Microsoft Word 中的选项对话框就是一个很好的例子）。

第三部分

深入学习 Qt

第 11 学时　使用布局管理器

第 12 学时　处理文件和目录

第 13 学时　处理文本和理解常规表达式

第 14 学时　学习使用容器类

第 15 学时　深入理解图形

第 16 学时　程序间通信

第 11 学时　使用布局管理器

到现在为止，我们都是通过向 QWidget::resize()和 QWidget::setGeometry()函数传递绝对坐标的方法来放置 Qt 对象。使用这种方法存在的问题是，当调整父窗口尺寸时，它可能变得太大或太小而不适合其中的子部件。从而使程序看起来不够美观，这时就需要使用布局管理器。

布局管理器是一套能够控制怎样和什么时候调整主窗口尺寸的 Qt 类。如果在布局管理器中注册你的部件，并且不指定它们的绝对位置，那么当用户调整父窗口尺寸时，布局管理器就会根据需要重新调整或移动子部件。这使应用程序更加容易使用，也使其界面更加美观。

当放置很多部件时，布局管理器是一个很好的助手。这一过程往往非常费时，而使用布局管理器，则能更快地完成它。

11.1　理解布局管理器

为了使你更好地理解布局管理器的概念（也称做几何位置管理或布局管理），我们现在来看一个实际的例子（尽管它非常简单）。

图 11-1 显示出的窗口具有两个 QPushButton 部件。

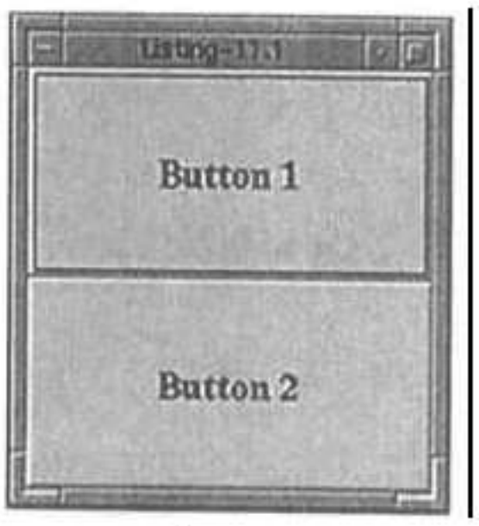


图 11-1　具有两个 QPushButton 部件的窗口，它没有连接到布局管理器

假如你想使窗口变大一点，其结果看起来将会与图 11-2 一样。

相反，如果在图 11-1 所示例子中使用布局管理器，调整窗口后的显示结果将如图 11-3 所示。

前面已经提到，当需要放置多个部件时，你也可以使用布局管理器。例如，如果你想放置 10 个 QPushButton 按钮，为每一个按钮输入绝对位置将耗费很多时间。相反，你可以创建按钮，但不输入它们的绝对位置，之后，使用布局管理器注册每一个对象。

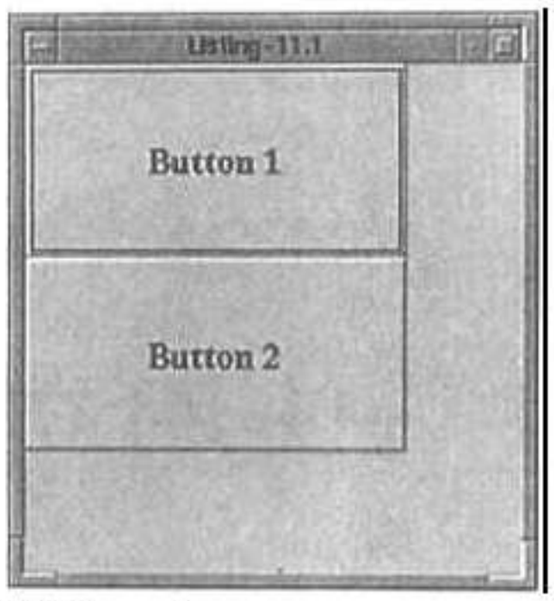


图 11-2　调整未使用布局管理器的图像。一般你会希望部件（这里为按钮）能够随主窗口移动或调整

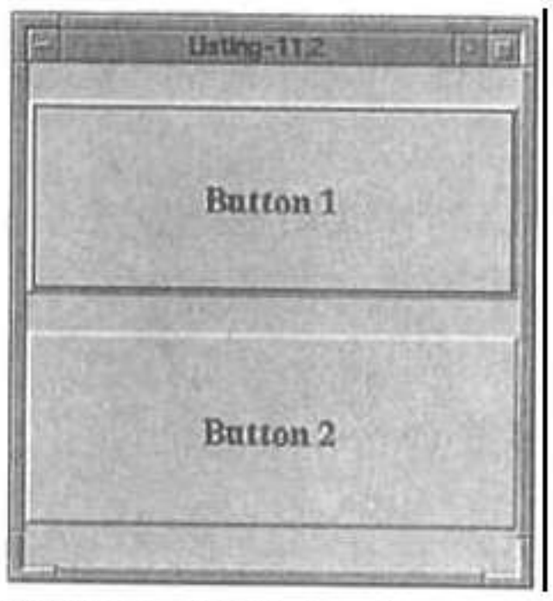


图 11-3　使用布局管理器控制按钮位置。当调整窗口时，按钮位置被改变以适应新的窗口尺寸

11.2　使用布局管理器

现在，你知道什么是布局管理器和怎样使用布局管理器。因此，我们来看怎样在程序中实现它们。

首先，你需要知道 Qt 中包含哪些布局管理器，以及怎样选择它们。表 11-1 列出 Qt 2.0 所包含的布局管理器及它们的作用。

表 11-1　Qt 布局管理器

布局管理器	描　述
QLayout	这是布局管理器的基类，所有其他布局管理器继承这个类。如果你想创建一个用户布局管理器，它应当基于 Qlayout
QGridLayout	当你想在网格中安排部件时，使用该布局管理器
QBoxLayout	这是 QHBoxLayout 和 QVBoxLayout 的基类。只有当在编译时刻无法确定是按列还是按行放置部件时才使用该类
QHBoxLayout	当按行放置部件时使用该类
QVBoxLayout	当按列放置部件时使用该类

上表中的第一个类是其他 4 个类的基类。几乎不需要使用这个类，因为混合使用 QGridLayout、QBoxLayout、QHBoxLayout 和 QVBoxLayout 能够解决大部分（而不是全部）布局问题（程序清单 11-5 给出一个例子，说明怎样混合使用这些布局管理器实行更复杂的

布局)。

11.2.1　按行和列安排部件

在“理解布局管理器”一节的例子中，按列放置两个 QPushButton 部件。在第一个例子中，按钮被手工放置。在第二个例子中，使用 QVBoxLayout。图 11-1 的源代码如程序清单 11-1 所示。

程序清单 11-1　　手工放置部件

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qpushbutton.h>
 4: #include <qfont.h>
 5:
 6: class MyWidget : public QWidget
 7: {
 8: public:
 9:     MyWidget();
10: };
11:
12: MyWidget::MyWidget()
13: {
14:     setMinimumSize( 200, 200 );
15:
16:     QPushButton *b1 = new QPushButton( "Button 1", this );
17:     b1->setGeometry( 0, 0, 200, 100 );
18:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
19:
20:     QPushButton *b2 = new QPushButton( "Button 2", this );
21:     b2->setGeometry( 0, 100, 200, 100 );
22:     b2->setFont( QFont( "Times", 18, QFont::Bold ) );
23: }
24:
25: int main( int argc, char **argv )
26: {
27:     QApplication a( argc, argv );
28:     MyWidget w;
29:     w.setGeometry( 100, 100, 200, 200 );
30:     a.setMainWidget( &w );
31:     w.show();
32:     return a.exec();
33: }
```

正如你所看到的，程序清单 11-1 没有给出任何新内容。但是，你一定要理解该代码，让我们来分析一下这个程序。首先，创建一个新类，MyWidget，它继承 QWidget 基类。之后，在 MyWidget 的唯一构造函数中创建两个按钮。最后，将 MyWidget 的最小尺寸设置为 200×200 像素。它就是如此简单！但是，如前所述，这个例子存在的问题是两个按钮，b1 和 b2，不能随着其父部件（MyWidget）的调整而调整/移动。在程序清单 11-2 中（如图 11-3 所示），这个问题被解决。

程序清单 11-2　　使用 QVBoxLayout 放置部件

```
1: #include <qapplication.h>
```

```
 2: #include <qwidget.h>
 3: #include <qpushbutton.h>
 4: #include <qfont.h>
 5: #include <qlayout.h>
 6:
 7: class MyWidget : public QWidget
 8: {
 9: public:
10:     MyWidget();
11: };
12:
13: MyWidget::MyWidget()
14: {
15:     setMinimumSize( 200, 200 );
16:
17:     QPushButton *b1 = new QPushButton( "Button 1", this );
18:     b1->setMinimumSize( 200, 100 );
19:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
20:
21:     QPushButton *b2 = new QPushButton( "Button 2", this );
22:     b2->setMinimumSize( 200, 100 );
23:     b2->setFont( QFont( "Times", 18, QFont::Bold ) );
24:
25:     QVBoxLayout *vbox = new QVBoxLayout( this );
26:     vbox->addWidget( b1 );
27:     vbox->addWidget( b2 );
28: }
29: int main( int argc, char **argv )
30: {
31:     QApplication a( argc, argv );
32:     MyWidget w;
33:     w.setGeometry( 100, 100, 200, 200 );
34:     a.setMainWidget( &w );
35:     w.show();
36:     return a.exec();
37: }
```

程序的这个版本使用布局管理器（vbox，基于 QVBoxLayout）放置部件，因此，看起来有一点差别。我们来逐个说明这些差别：

● 首先，它包含 Qt 库中一个新的头文件，qlayout.h。这个文件定义 Qt 布局管理器类。在这里，包含头文件 qlayout.h 是为了访问 QVBoxLayout 类；

● 你可能注意到的第二个差别是不用 QWidget::setGeometry()来改变按钮的大小和位置。相反，它使用 QWidget::setMinimumSize()设置按钮的最小尺寸（两种情况下都为 200×100 像素）。当程序首次运行时将使用这一尺寸显示按钮，但其大小不是绝对的；

● 最后，一个全新对象被添加到应用程序。它是 QVBoxLayout 类对象，叫做 vbox。这是程序中目前的布局管理器。为了使 vbox 知道按钮（b1 和 b2），并使它对它们进行布置，这些按钮必须在布局管理器中注册。这在第 26 和 27 行中使用 QVBoxLayout::addWidget() 函数实现。当你使用布局管理器时，所有的部件都使用这种方法进行注册。



不只是 QPushButton 部件能够被布局管理器注册和控制。事实上，使用布局管理器能够控制所有 Qt 部件的布局。

这并不难，不是吗？如果你想按行方式安排部件，只需使用 QHBoxLayout 代替 QVBoxLayout 即可。

11.2.2　QGridLayout

如果你想以网格状形式安排 Qt 部件，则应该使用 QGridLayout 类（使用 QVBoxLayout 和 QHBoxLayout 也能创建每一行和列，但是，使用 QGridLayout 更容易）。我们来看一个例子，它使用 QGridLayout 将 6 个 QPushButton 对象放置为 3 行 2 列。这个例子的源代码如程序清单 11-3 所示：

程序清单 11-3　　使用 QGridLayout 放置部件

```
#include <qapplication.h>
#include <qwidget.h>
#include <qpushbutton.h>
#include <qfont.h>
#include <qlayout.h>

class MyWidget : public QWidget
{
public:
    MyWidget();
};

MyWidget::MyWidget()
{
    setMinimumSize( 200, 200 );

    QPushButton *b1 = new QPushButton( "Button 1", this );
    b1->setMinimumSize( 200, 100 );
    b1->setFont( QFont( "Times", 18, QFont::Bold ) );

    QPushButton *b2 = new QPushButton( "Button 2", this );
    b2->setMinimumSize( 200, 100 );
    b2->setFont( QFont( "Times", 18, QFont::Bold ) );

    QPushButton *b3 = new QPushButton( "Button 3", this );
    b3->setMinimumSize( 200, 100 );
    b3->setFont( QFont( "Times", 18, QFont::Bold ) );

    QPushButton *b4 = new QPushButton( "Button 4", this );
    b4->setMinimumSize( 200, 100 );
    b4->setFont( QFont( "Times", 18, QFont::Bold ) );

    QPushButton *b5 = new QPushButton( "Button 5", this );
    b5->setMinimumSize( 200, 100 );
    b5->setFont( QFont( "Times", 18, QFont::Bold ) );

    QPushButton *b6 = new QPushButton( "Button 6", this );
    b6->setMinimumSize( 200, 100 );
    b6->setFont( QFont( "Times", 18, QFont::Bold ) );

    QGridLayout *grid = new QGridLayout( this, 3, 2 );
    grid->addWidget( b1, 0, 0 );
    grid->addWidget( b2, 1, 0 );
```

```
44:     grid->addWidget( b3, 2, 0 );
45:     grid->addWidget( b4, 0, 1 );
46:     grid->addWidget( b5, 1, 1 );
47:     grid->addWidget( b6, 2, 1 );
48: }
49:
50: int main( int argc, char **argv )
51: {
52:     QApplication a( argc, argv );
53:     MyWidget w;
54:     w.setGeometry( 100, 100, 200, 200 );
55:     a.setMainWidget( &w );
56:     w.show();
57:     return a.exec();
58: }
```

这个例子结果如图 11-4 所示。

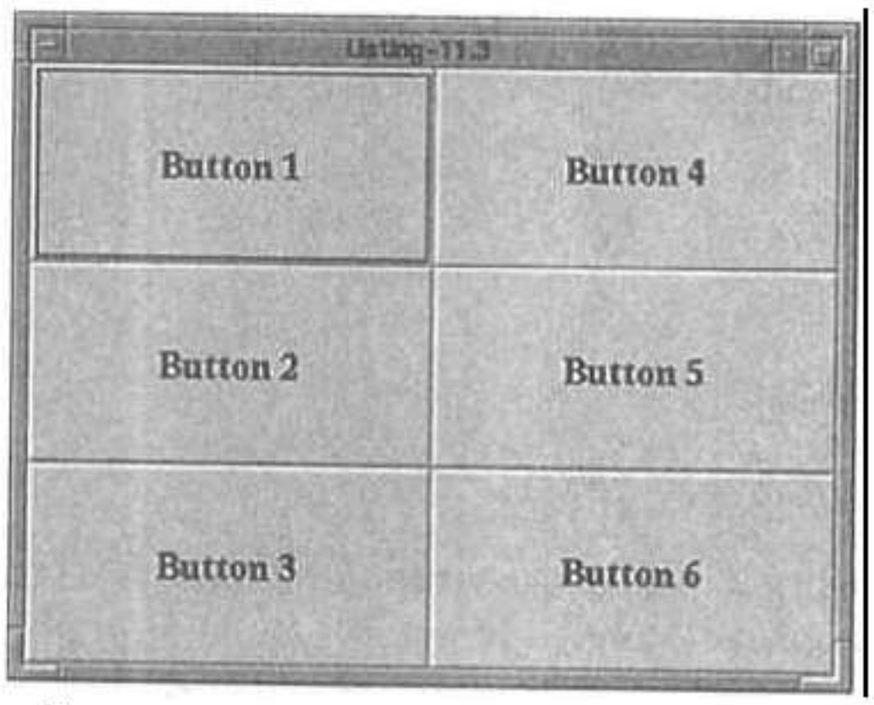


图 11-4 使用 QGridLayout 按网格状放置 6 个按钮

这里，用 QGridLayout 类控制按钮。这个类的创建和使用语法与 QVBoxLayout 和 QHBoxLayout 有一点不同。首先，需要传递给 QGridLayout 构造函数 3 个参数而不是 1 个。这在该行中实现：

```
QGridLayout *grid = new QGridLayout( this, 3, 2 ) ;
```

第 1 个参数，this，与通常一样，它是一个指向未创建的 MyWidget 对象的指针。但是，第 2 个和第 3 个参数是你不熟悉的。第 2 个参数，3，指出网格中应该有多少行。第 3 个参数，2，指出网格中应该有多少列。在这里，有 3 行 2 列。

其次，与使用 QVBoxLayout 和 QHBoxLayout 一样，用 QGridLayout::addWidget()函数向布局管理器（网格）添加部件（这里为按钮）。正如你所看到的，这里增加两个参数。除 this 指针作为第一个参数传递外，还需告诉 QGridLayout::addWidget()在哪一行和列上插入按钮。来看下面一行：

```
grid->addWidget( b1, 0, 0 ) ;
```

这里，向网格中的第一行和第一列插入一个对象（b1）。这表示图 11-4 中左上角的按钮。因此，在程序清单 11-3 的例子中，（0，1）代表按钮 4，（2，1）代表按钮 6，等等。图 11-5 用一个更好的视图显示这个例子。

现在，你知道怎样使用 QGridLayout 创建部件网格。关于这一问题的更多信息，请参看

Qt Reference Document 和这一学时前面一节“理解布局管理器”。

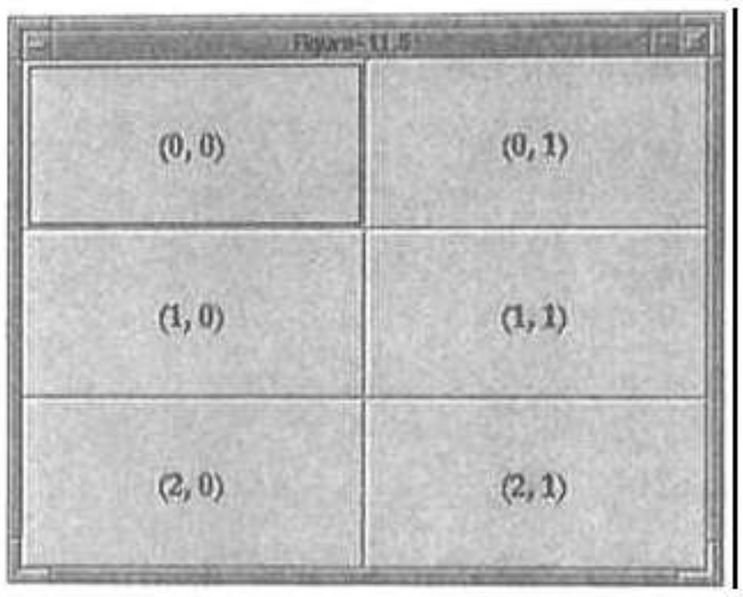


图 11-5　怎样标识网格单元

11.3　理解嵌套布局管理器

如果熟悉 C 程序设计（或其他任何程序设计语言），你可能已经嵌套使用过程序的某些功能以获得一些其他功能。例如，在 C 语言中嵌套 if 语句就很常用的。基本上是把 if 语句放置在其他 if 语句内，使用这种方式能够更好地控制程序流程。使用布局管理器时，嵌套也是很有用的。

例如，假如想修改程序清单 11-3 中的代码，使其左边一列有 3 行，右边一列只有两行。用嵌套布局管理器，就能够很容易地实现这一点。

程序清单 11-4 解决这一问题。其图形显示如图 11-6 所示。

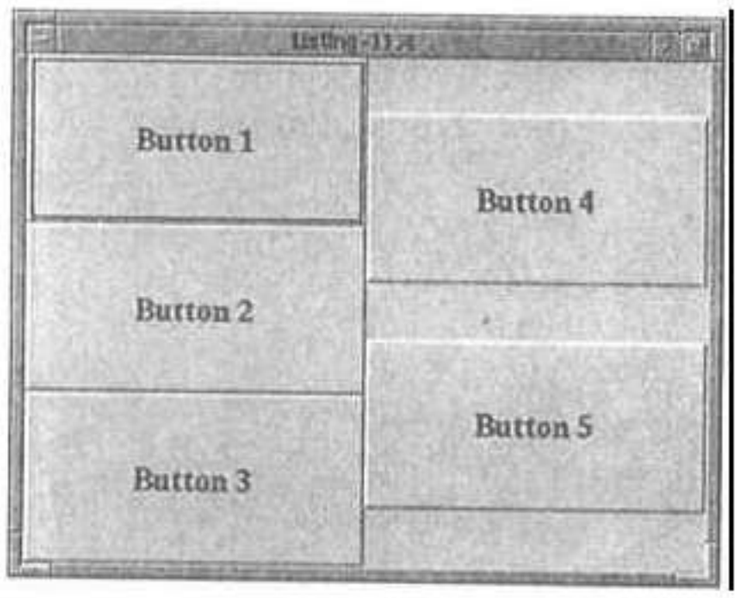


图 11-6　使用嵌套管理器放置按钮



当然，使用手工放置部件方式也能很容易地实现。但是，之后你不能得到布局管理器的功能。

程序清单 11-4　使用嵌套管理器放置部件

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qpushbutton.h>
```

```
 4: #include <qfont.h>
 5: #include <qlayout.h>
 6:
 7: class MyWidget : public QWidget
 8: {
 9: public:
10:     MyWidget();
11: };
12:
13: MyWidget::MyWidget()
14: {
15:     setMinimumSize( 200, 200 );
16:
17:     QPushButton *b1 = new QPushButton( "Button 1", this );
18:     b1->setMinimumSize( 200, 100 );
19:     b1->setFont( QFont( "Times", 18, QFont::Bold ) );
20:
21:     QPushButton *b2 = new QPushButton( "Button 2", this );
22:     b2->setMinimumSize( 200, 100 );
23:     b2->setFont( QFont( "Times", 18, QFont::Bold ) );
24:
25:     QPushButton *b3 = new QPushButton( "Button 3", this );
26:     b3->setMinimumSize( 200, 100 );
27:     b3->setFont( QFont( "Times", 18, QFont::Bold ) );
28:
29:     QPushButton *b4 = new QPushButton( "Button 4", this );
30:     b4->setMinimumSize( 200, 100 );
31:     b4->setFont( QFont( "Times", 18, QFont::Bold ) );
32:
33:     QPushButton *b5 = new QPushButton( "Button 5", this );
34:     b5->setMinimumSize( 200, 100 );
35:     b5->setFont( QFont( "Times", 18, QFont::Bold ) );
36:
37:     QHBoxLayout *hbox = new QHBoxLayout( this );
38:     QVBoxLayout *vbox1 = new QVBoxLayout();
39:     QVBoxLayout *vbox2 = new QVBoxLayout();
40:
41:     hbox->addLayout( vbox1 );
42:     hbox->addLayout( vbox2 );
43:
44:     vbox1->addWidget( b1 );
45:     vbox1->addWidget( b2 );
46:     vbox1->addWidget( b3 );
47:
48:     vbox2->addWidget( b4 );
49:     vbox2->addWidget( b5 );
50: }
51:
52: int main( int argc, char **argv )
53: {
54:     QApplication a( argc, argv );
55:     MyWidget w;
56:     w.setGeometry( 100, 100, 200, 200 );
57:     a.setMainWidget( &w );
58:     w.show();
59:     return a.exec();
60: }
```

你可能不熟悉程序清单 11-4 中的部分代码。但这些新代码并不难理解。在第 37、38 和 39 行，创建 3 个布局管理器——两个 QVBoxLayout（vbox1 和 vbox2）和一个 QHBoxLayout（hbox）。对于两个 QVBoxLayout 对象，它们没有成为父部件（见第 41 和 42 行）。在这两行中，两个 QVBoxLayout 对象被插入到 QHBoxLayout 布局管理器中，这使 vbox1 和 vbox2 成为 hbox 的子部件。注意，这里没有使用 QLayout::addWidget()函数。取而代之的是 QLayout::addLayout()函数。在第 44 行到第 49 行，向 vbox1 和 vbox2 插入 5 个 QPushButton 对象。



有一些真正有用的函数能够根据屏幕大小和窗口大小来动态放置部件：

- 调用 QApplication::desktop()->width()和 QApplication:: desktop()->height()能够读取当前运行程序的屏幕宽度和高度；
- 从类构造函数中调用 this->width()和 this->height()能够读取部件的宽度和高度（很可能是主窗口）。

图 11-7 以图形方式解释程序清单 11-4 中程序。

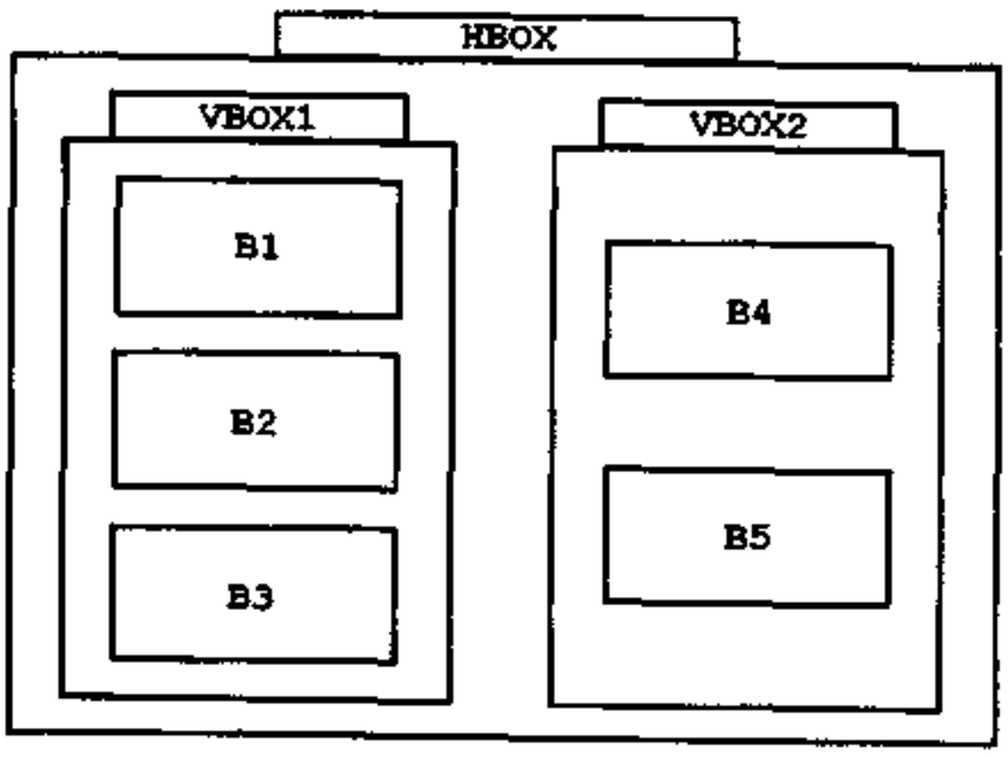


图 11-7　程序清单 11-4 中的布局管理器的组织方式

一个实例

前一个使用 QPushButton 对象的例子使你初步了解了布局管理器。但是，你在程序中很可能不只是包含几个按钮，它们可能也要包含其他部件。现在我们来看一个更实际的例子，这个程序的源代码如程序清单 11-5 所示：

程序清单 11-5　　怎样使用布局管理器实例

```
1: #include <qapplication.h>
2: #include <qlabel.h>
3: #include <qcolor.h>
4: #include <qpushbutton.h>
5: #include <qlayout.h>
6: #include <qlineedit.h>
7: #include <qmultilineedit.h>
8: #include <qmenubar.h>
```

```
 9: #include <qpopupmenu.h>
10:
11: class RLExample : public QWidget
12: {
13: public:
14:     RLExample();
15:     ~RLExample();
16: };
17;
18: RLExample::RLExample()
19: {
20:     QVBoxLayout *topLayout = new QVBoxLayout( this, 5 );
21:
22:     QMenuBar *menubar = new QMenuBar( this );
23:     menubar->setSeparator( QMenuBar::InWindowsStyle );
24:     QPopupMenu* popup;
25:     popup = new QPopupMenu;
26:     popup->insertItem( "&Quit", qApp, SLOT(quit()) );
27:     menubar->insertItem( "&File", popup );
28:
29:     topLayout->setMenuBar( menubar );
30:
31:     QHBoxLayout *buttons = new QHBoxLayout( topLayout );
32:     int i;
33:
34:     for ( i = 1; i <= 4; i++ )
35:     {
36:     QPushButton *but = new QPushButton( this );
37:     QString s;
38:     s.sprintf( "Button %d", i );
39:     but->setText( s );
40:
41:     buttons->addWidget( but, 10 ); used
42:     }
43:
44:     QHBoxLayout *buttons2 = new QHBoxLayout( topLayout );
45:     QPushButton *but = new QPushButton( "Button five", this );
46:     buttons2->addWidget( but );
47:     but = new QPushButton( "Button 6", this );
48:     buttons2->addWidget( but );
49:     buttons2->addStretch( 10 );
50:
51:     QMultiLineEdit *bigWidget = new QMultiLineEdit( this );
52:     bigWidget->setText( "This widget will get all the remaining space" );
53:     bigWidget->setFrameStyle( QFrame::Panel | QFrame::Plain );
54:     topLayout->addWidget( bigWidget );
55:
56:     const int numRows = 3;
57:     const int labelCol = 0;
58:     const int linedCol = 1;
59:     const int multiCol = 2;
60:
61:     QGridLayout *grid = new QGridLayout( topLayout, 0, 0, 10 );
62:     int row;
63:
64:     for ( row = 0; row < numRows; row++ )
65:     {
66:     QLineEdit *ed = new QLineEdit( this );
```

```
67:     grid->addWidget( ed, row, linedCol );
68:
69:     QLabel *label = new QLabel( this );
70:     QString s;
71:     s.sprintf( "Line &%d", row+1 );
72:     label->setText( s );
73:     grid->addWidget( label, row, labelCol );
74:     label->setBuddy( ed );
75:     }
76:
77:     QMultiLineEdit *med = new QMultiLineEdit( this );
78:     grid->addMultiCellWidget( med, 0, -1, multiCol, multiCol );
79:     grid->setColStretch( linedCol, 10 );
80:     grid->setColStretch( multiCol, 20 );
81:
82:     QLabel *sb = new QLabel( this );
83:     sb->setText("Let's pretend this is a status bar");
84:     sb->setFrameStyle( QFrame::Panel | QFrame::Sunken );
85:     sb->setFixedHeight( sb->sizeHint().height() );
86:     sb->setAlignment( AlignVCenter | AlignLeft ); used
87:
88:     topLayout->addWidget( sb );
89: }
90:
91: RLExample::-RLExample()
92: {
93:     // All child widgets are deleted by Qt.
94: }
95:
96: int main( int argc, char **argv )
97: {
98:     QApplication a( argc, argv );
99:     RLExample *w = new RLExample;
100:    a.setMainWidget(w);
101:    w->show();
102:    return a.exec();
103: }
```

建议你深入研究这段代码，遇到不熟悉的问题时要毫不犹豫地去查阅 Qt Reference Document。注意，这个例子中的部件不是都具有相同的伸展系数。伸展系数不同时，它们被调整的尺寸也不相同。例如，QMultiLineEdit 部件 bigwidget 将获得所有垂直方向上的新增空间，而其他部件则共享水平方向上的新增空间。程序清单 11-5 的图形解释如图 11-8 所示，其执行结果如图 11-9 所示。

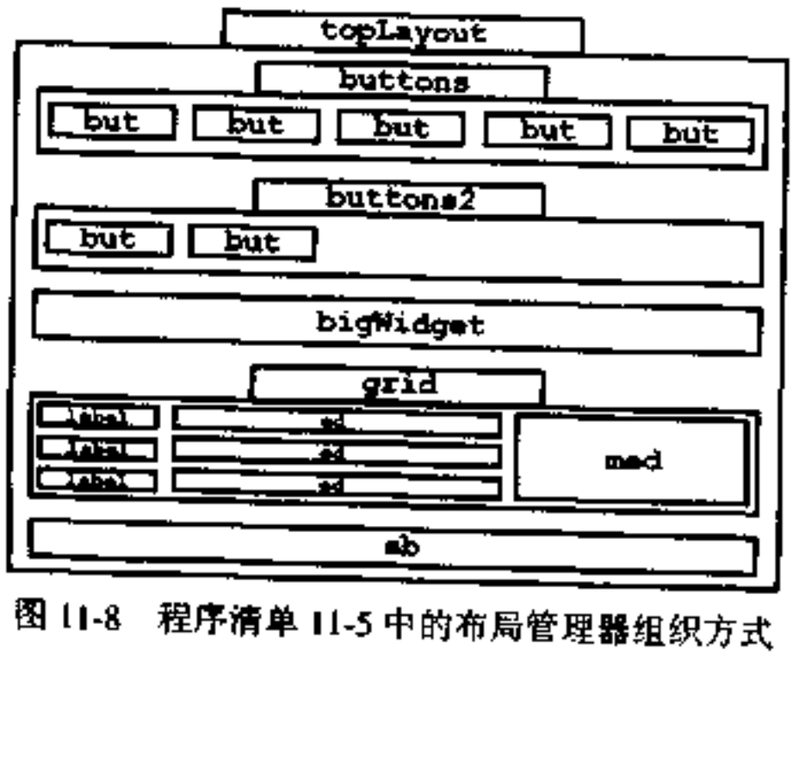


图 11-8　程序清单 11-5 中的布局管理器组织方式

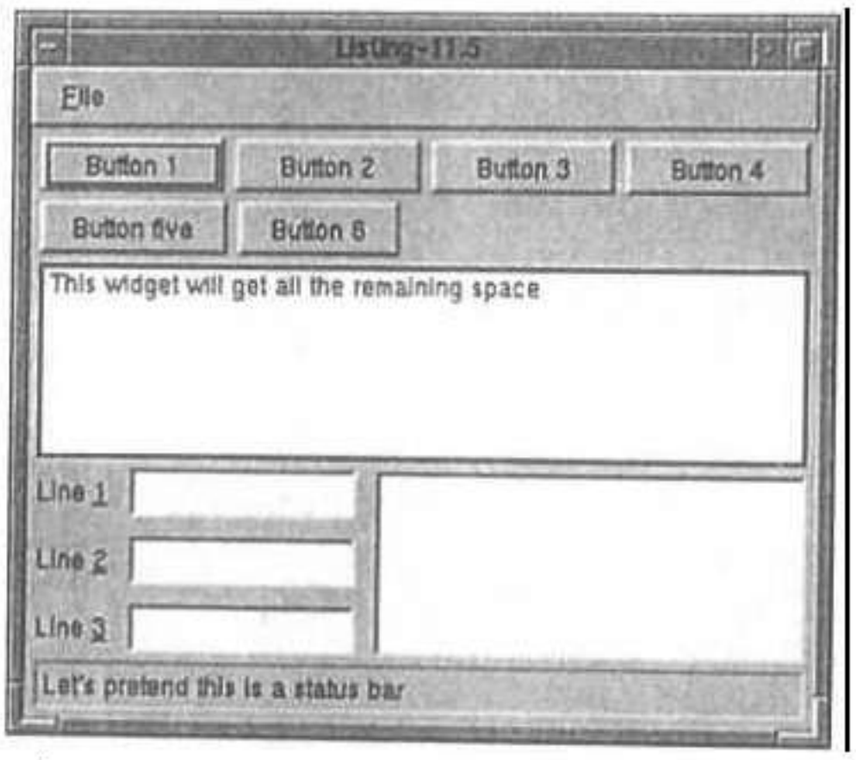


图 11-9　程序清单 11-5 运行结果，如你所看到的，布局管理器也能够安排除按钮以外的其他部件

现在，你知道了嵌套布局管理器的工作方式。你可以用无数种方法组合使用不同的布局管理器以实现你的布局目标。



在某些情况下，即使嵌套布局管理器也不能完成你的工作。这时你有两种选择：进行手工布局或者创建你自己的用户布局管理器。这在 Qt Reference Document 的 customlayout.html 文件中介绍。

11.4　小　结

布局管理器是 Qt 库非常重要的一部分。在很多情况下，使用布局管理器代替手工布局能够节省大量时间。你可能认为坚持使用常用（更耗时）的部件放置方法更容易，但是，你应该更多地使用布局管理器。

当然，这一课并没有介绍布局管理器的所有内容。但是，Qt Reference Document 覆盖到了，你应该从其中查阅更多信息。从中你将发现很多在这一课中没有讲到的功能。

11.5　问题与答案

问：Qt 2.0 和 Qt 1.4x 中的布局管理器使用有什么差别？

答：事实上有。在旧的 Qt 1.4x 版本中，你必须调用 QLayout::Activate()来激活布局管理器。

问：我设计一个很特殊的布局，布局管理器无法实现它们。我应该怎样做？

答：有两种选择：可以通过基于 QLayout 类创建一个新类来创建你自己的布局管理器或是重新实现 QWidget::resizeEvent()函数。这两种方法都在 Qt Reference Document 中讲到。

问：哪些 Qt 部件能够被布局管理器控制？

答：所有的 Qt 部件都能够被布局管理器控制。为了清楚起见，这一课主要使用

QPushButton 部件。记住当向布局管理器中插入部件时要使用正确的函数。

11.6 作　业

希望你完成下面的问题和练习，这有助于你牢记本课所学内容。完成这些问题也有助于你理解布局管理器的工作方式。

11.6.1 测验

1. 什么是布局管理器？
2. 布局管理器能帮助你做什么？
3. QVBoxLayout 怎样组织子部件？
4. QHBoxLayout 怎样组织子部件？
5. QGridLayout 怎样组织子部件？
6. 在 QGridLayout 布局管理器中哪两个数代表左上角的部件？
7. 什么时候需要使用嵌套布局管理器？
8. 什么时候必须调用 QLayout::Avtivate()函数？

11.6.2 练习

1. 修改程序清单 11-2 中的程序，使它按行而不是按列方式组织两个按钮（所修改的字符数不允许超过两个）。
2. 编写一个程序，使用 QGridLayout 按两行 3 列方式组织 6 个按钮。
3. 编写一个使用嵌套布局管理器的程序。它应该在 3 行 2 列中放置 5 个按钮。第一和第 3 行有两个按钮，而第二行（中间一行）只有一个。

第 12 学时　处理文件和目录

你是否想象过文件和目录在计算机中所起的重要作用？如果没有这两个元素，将不可能在计算机上存储任何东西，计算机也不会像今天这样有用。或许这听起来有点愚蠢，但是，过去一段时间计算机就没有足够的存储空间（像今天的计算机这样）。那时，在计算机关闭时存储信息就总是一个问题。人们一直在不倦地努力克服它。

幸运的是，技术的发展已今非昔比。因此，可以完全想象你的硬盘上至少有一些空闲空间供使用。

在这一课，将学习怎样使用硬盘驱动器（也被称做辅助内存）来存储和阅读信息。换句话说，将学习怎样使用文件和目录。如果你不想使信息在退出程序和/或关闭计算机后就消失的话，这将是一个很重要的内容。

以前，你可能在 C 程序中使用过文件和目录，如果是这样的话，你很可能使用过与操作系统相关的文件和目录函数。但是，如果这样做，应用程序将完全不能移植，它将不能在你所用平台以外的平台下使用。相反，如果使用 Qt 类，程序能够在所有其他 Qt 平台上使用。更重要的是，即使移植对你来说不是一个很重要的问题（并且以后也不会发生），使用 Qt 类也会使文件和目录的使用更加简单。

12.1　使用 Qt 类读取文件

或许你所执行的最简单的文件操作就是读取文件。例如，如果想向用户显示帮助文件，那么，将帮助文本存入一个文件中要比在程序中实现整个文本要方便得多。

现在，你将看到一个怎样实现这一操作的例子。这里使用 Qt 类打开 Qt 所带的 README 文件，并创建一个指向它的文本流，然后用 QMultiLineEdit 对象显示它。程序清单 12-1 说明这个例子的实现过程：

程序清单 12-1　　读取文本文件并在屏幕上显示

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qfile.h>
4: #include <qtextstream.h>
5: #include <qstring.h>
6: #include <qmultilineedit.h>
7:
```

```
 8: class MyMainWindow : public QWidget
 9: {
10: public:
11:     MyMainWindow();
12: private:
13:     QMultiLineEdit *medit;
14: };
15:
16: MyMainWindow::MyMainWindow()
17: {
18:     setGeometry( 100, 100, 480, 400 );
19:
20:     //Create the QMultiLineEdit object in
21:     //which we will show the text file:
22:     medit = new QMultiLineEdit( this );
23:     medit->setGeometry( 10, 10, 460, 380 );
24:     medit->setReadOnly( TRUE );
25:
26:     //Here, we assign the object file myFile
27:     //to the Qt README file.
28:     QFile myFile( "/usr/local/qt/README" );
29:
30:     //Set the mode to read-only:
31:     myFile.open( IO_ReadOnly );
32:
33:     //Create a text stream for the QFile object:
34:     QTextStream myStream( &myFile );
35:
36:     //Create a QString object and use the text
37:     //stream to read the file contents into it:
38:     QString myString;
39:
40:     //Until we have reached the end of the
41:     //file, read one line at a time:
42:     while( myStream.atEnd() == 0 )
43:     {
44:         //Read one line from the file:
45:         myString = myStream.readLine();
46:
47:         //Insert that line to the
48:         //QMultiLineEdit object:
49:         medit->insertLine( myString );
50:     }
51:
52:     //Close the connection to the file:
53:     myFile.close();
54: }
55:
56: void main( int argc, char **argv )
57: {
58:     QApplication a(argc, argv);
59:     MyMainWindow w;
60:     a.setMainWidget( &w );
61:     w.show();
62:     a.exec();
63: }
```

这里，使用标准模式读取文件。首先，创建一个代表文件的 QFile 对象（第 28 行）。之

后，设置 QFile 对象模式（这里为只读模式）（在第 31 行）。再创建一个 QTextStream 对象（myStream）（第 34 行），这个对象从文件中读取文本。之后启动 while 循环（第 42 行），一次从文件中读取一行，并将它输出到 QMultiLineEdit 对象。当到达文件尾部时，停止 while 循环。最后，调用 QFile::close()函数关闭文件（第 53 行）。



如果在打开一个文件时出现问题，这可能是因为你不具有该模式下的打开权限。在前一个例子中，如果你没有将模式设置为只读方式，很可能不能打开文件，因为默认打开模式给你的权限比你在系统上实际具有的要多。
但是，在 Windows 95 和 98 中则不会经常出现这个问题，因为它不具有这样严格的权限。



在这个例子中，打开一个文本文件。但是，在某些时候可能需要以二进制方式打开文件。在这种情况下，你需要使用 QDataStream，而不是 QTextStream。
如果你使用 QDataStream 写入二进制文件，这些文件在所有 Qt 平台下都能移植。

该程序运行结果如图 12-1 所示。

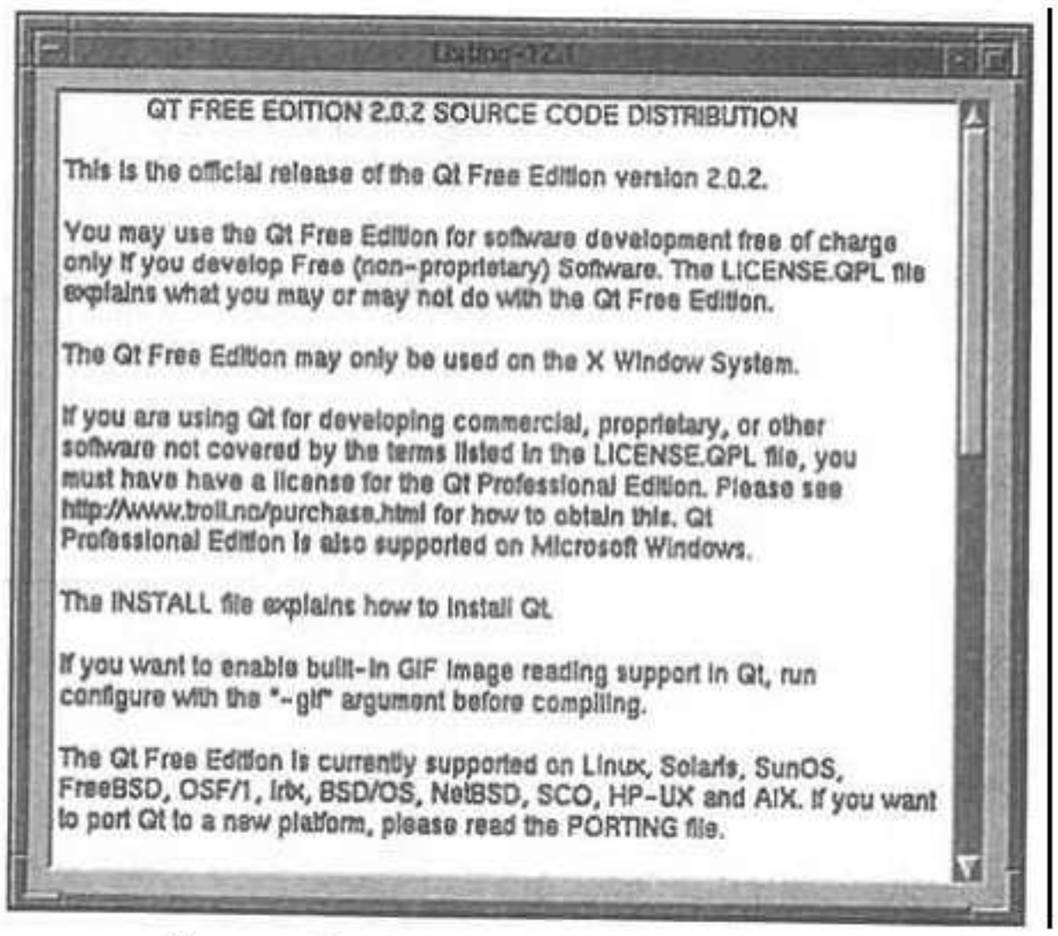


图 12-1　用 QMulitLineEdit 对象显示文本文件

前面已经讲过，程序中经常需要设置文件的打开模式。表 12-1 列出了所有打开模式。

表 12-1　　文件打开模式

模　式	解　释
IO_Raw	以原始模式（不使用缓冲区）打开文件。你可能很少使用这种模式
IO_ReadOnly	这种模式在程序清单 12-1 中已经使用。在这种模式下，你只能读取文件，而不能写入

续表

模　式	解　释
IO_WriteOnly	在这种模式下，你只能写入文件，而不能读取
IO_ReadWrite	以读写模式打开文件。使用这种模式能够同时读取和写入文件（与 IO_ReadOnly \| IO_WriteOnly 相同）
IO_Append	这种模式将文件指针设置到文件尾部。这意味着当向文件中写入内容时，它将被添加到文件的尾部。当你要向文件中添加文本而又不删除其当前内容时，这种模式就很有用
IO_Truncate	在这种模式下，如果文件存在它将被截断
IO_Translate	这种模式打开 MS-DOS、Windows 和 OS/2 系统的回车/换行转换功能

QTextStream 定义了常用的 C++流操作符（<<和>>）。因此，如果你习惯使用这些操作符，则可以使用它们读写文件。

另外几个有趣的 QTextStream 函数，其中包括 QTextStream::read() 和 QtextStreamsetEncoding()。QTextStream::readLine() 函数每次只读取一行，而 QTextStream::read()函数读取整个流。使用 QtextStreamsetEncoding()函数可以设置文本字符编码。

QFile 还有一些对你来说可能很有用的函数。例如，QFile::exists()函数，当文件存在时它返回布尔值 TRUE，而当文件不存在时则返回布尔值 FALSE。QFile::remove()删除当前文件。当删除成功时它返回 TRUE，否则返回 FALSE。使用 QFile::setName()可以设置文件名。调用 QFile::name()函数能够读取 QFile::setName()所设置的文件名。查看 Qt Reference Document 可以了解所有函数列表。

12.2　使用 Qt 类读取目录

在这一节将学习怎样使用 QDir 类。用它读取目录内容并将读取结果输出到一个 QMultiLineEdit 对象。在程序清单 12-2 中你将看到，使用 Qt 类非常容易实现这一功能——需要编写的代码不超过几行。

程序清单 12-2　　读取目录内容

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qmultilineedit.h>
4: #include <qdir.h>
5:
6: class MyMainWindow : public QWidget
7: {
8: public:
9:     MyMainWindow();
10: private:
11:     QMultiLineEdit *medit;
12: };
13:
14: MyMainWindow::MyMainWindow()
15: {
16:     resize( 170, 400 );
```

```
17:
18:     //Create the QMultiLineEdit object which
19:     //will display the files:
20:     medit = new QMultiLineEdit( this );
21:     medit->setGeometry( 10, 10, 150, 380 );
22:     medit->setReadOnly( TRUE );
23:
24:     //Create a QDir object for the /etc
25:     //directory:
26:     QDir myDir( "/etc" );
27:
28:     //Write each file name to the QMultiLineEdit
29:     //object on a line of its own:
30:     for( int i = 0; i < myDir.count(); i++ )
31:     {
32:         medit->insertLine( myDir[i] );
33:     }
34: }
35:
36: void main( int argc, char **argv )
37: {
38:     QApplication a(argc, argv);
39:     MyMainWindow w;
40:     a.setMainWidget( &w );
41:     w.show();
42:     a.exec();
43: }
```

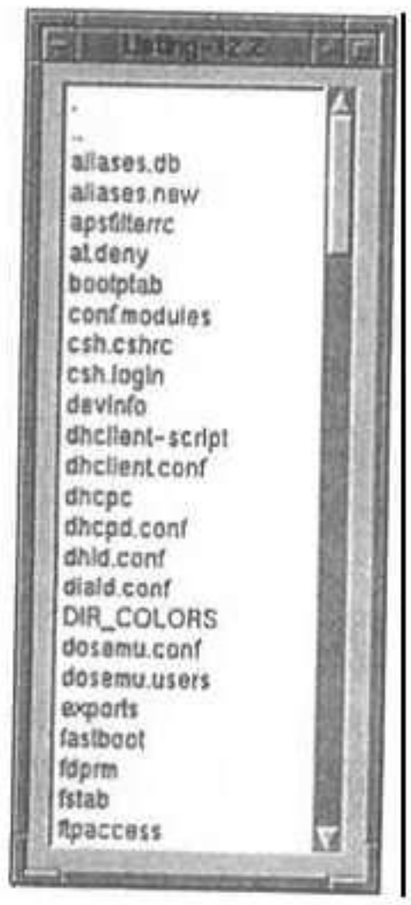


图 12-2　用 QMulitLineEdit 对象显示目录内容（这里为 Linux 文件系统中的/etc 目录）

这里所用的方法与读取文件不同。当创建 QDir 对象时（第 26 行），你不仅创建一个描述那个目录的对象，而且还读取整个目录中的内容，使用 QDir 对象的索引操作符[]能够很容易地访问这些内容。在前一个程序清单中，使用 for 循环逐行读取（第 32 行）目录中的所有内容。注意，这里是怎样使用 QDir::count()函数（第 30 行）读取目录中的内容。图 12-2 为该程序显示结果。

事实上，QDir 具有很多函数，它们能够执行不同类型的目录操作。表 12-2 列出其中一些最常用的函数。

表 12-2　　QDir 成员函数

函　数	描　述
QDir::setPath()	使用这个函数设置需要处理的目录路径。在程序清单 12-2 中，路径通过构造函数来定义
QDir::*path*()	返回当前所选目录的路径
QDir::absPath()	返回绝对路径——也就是以/（根目录）开头的路径
QDir::canonicalPath()	返回没有任何符号连接的路径（也就是真实路径）
QDir::dirName()	只返回目录名称（不是到目录的整个路径）
QDir::filePath ()	返回文件参数的路径
QDir:: absFilePath()	返回文件参数的绝对路径
QDir::cd()	与 cd 命令一样，切换到参数指定的目录名
QDir::cdup ()	在目录树中向上移动一级
QDir::setFilter ()	如果只想看到目录中某一类或几类项目，可以使用这个函数设置过滤器。然后通过 QDir::entryList()函数访问这个被过滤的列表。详细内容请参考 Qt Reference Document
QDir::mkdir ()	使用参数中指定的名称创建目录
QDir::rmdir ()	删除参数所指定的目录
QDir::exists ()	如果目录存在则返回 TRUE，否则，返回 FALSE。如果用文件名作为参数，函数将检查指定文件是否存在
QDir::remove ()	这个函数删除在参数中所指定的文件
QDir::rename ()	使用这个函数能够重命名文件。第一个参数为文件名，第二个参数为新的文件名称

使用 QDir 时需要记住的另一件事情就是不必将目录路径从 UNIX 格式转换为 Windows 格式。如果使用/（UNIX 格式）来分隔目录名，在 Windows 下编译时 Qt 将负责转换。

QDir 是一个很有用的类。从我个人来讲，我已经在除 Qt 程序外的其他程序中多次使用这个类，因为它具有很强大的功能。尽管标准的 C++类也能够实现 QDir 类的功能，但是 Qt 类更加容易使用这一事实使它优于所有其他同类产品（至少是所有我测试过的同类产品）。这方面一个很好的例子就是怎样使用[]操作符来遍历整个目录。使用 C++标准函数就难以实现，尽管它也完全能够遍历一个目录。

12.3　使用 Qt 类读取文件信息

如果将所有文件比作人，那么，文件属性则相当于文件的个性。属性描述文件外观、它能做什么以及不能做什么等等。需要了解文件属性才能知道应该怎样去使用它。例如，在了解每个雇员能够做什么之前，雇主无法决定雇佣哪个雇员，以及这些未来的雇员能否适合其工作小组。同样，你需要了解一个文件的大小、以及它是否是可读、可写和可执行文件。

为此，需要使用 Qt 类 QFileInfo。使用这个类，能够很容易地读取所需文件信息。只需创建一个 QFileInfo 对象，并将所要了解的文件名称字符串传递给它（也可以将一个 QFile 对象传递给它）。这样做之后，就能够调用各种各样的 QFileInfo 成员函数来读取文件属性。

如果一个条目（文件、目录或符号连接）具有函数名所指属性，所有这些函数则返回 TRUE。否则，返回 FALSE。程序清单 12-3 给出一个例子，说明 QFileInfo 的用法：

程序清单 12-3　　读取目录中的项目属性

```
#include <qapplication.h>
#include <qwidget.h>
#include <qmultilineedit.h>
#include <qdir.h>
#include <qfileinfo.h>

class MyMainWindow : public QWidget
{
public:
  MyMainWindow();
private:
  QMultiLineEdit *medit;
  QFileInfo *finfo;
};

MyMainWindow::MyMainWindow()
{
  resize( 180, 400 );

  //Create the QMultiLineEdit object which
  //will display the files:
  medit = new QMultiLineEdit( this );
  medit->setGeometry( 10, 10, 160, 380 );
  medit->setReadOnly( TRUE );

  QDir myDir( "/" );

  //We create an int variable to control
  //on which line in the QMultiLineEdit
  //object we are inserting:
  int line;

  //This for-loop will investigate each item
  //in the directory myDir. Each item will be
  //processed through the if-statements to
  //determine which attributes the item has.
  for( int i = 0; i < myDir.count(); i++ )
    {
      finfo = new QFileInfo( myDir[i] );

      medit->insertLine( finfo->filePath(), line );
      line++;

      if( finfo->isFile() == TRUE )
    {
      medit->insertLine( "is a file.", line );
      line++;
    }

      if( finfo->isDir() == TRUE )
```

```
51:     {
52:       medit->insertLine( "is a directory.", line );
53:       line++;
54:     }
55:
56:       if( finfo->isSymLink() == TRUE )
57:     {
58:       medit->insertLine( "is a symbolic link.", line );
59:       line++;
60:     }
61:
62:       if( finfo->isReadable() == TRUE )
63:     {
64:       medit->insertLine( "is readable.", line );
65:       line++;
66:     }
67:       else
68:     {
69:       medit->insertLine( "is not readable.", line );
70:       line++;
71:     }
72:
73:       if( finfo->isWritable() == TRUE )
74:     {
75:       medit->insertLine( "is writable.", line );
76:       line++;
77:     }
78:       else
79:     {
80:       medit->insertLine( "is not writable.", line );
81:       line++;
82:     }
83:
84:       if( finfo->isExecutable() == TRUE )
85:     {
86:       medit->insertLine( "is executable.", line );
87:       line++;
88:     }
89:       else
90:     {
91:       medit->insertLine( "is not executable", line );
92:       line++;
93:     }
94:
95:       medit->insertLine( " " );
96:       line++;
97:     }
98: }
99:
100: void main( int argc, char **argv )
101: {
102:     QApplication a(argc, argv);
103:     MyMainWindow w;
104:     a.setMainWidget( &w );
105:     w.show();
106:     a.exec();
107: }
```

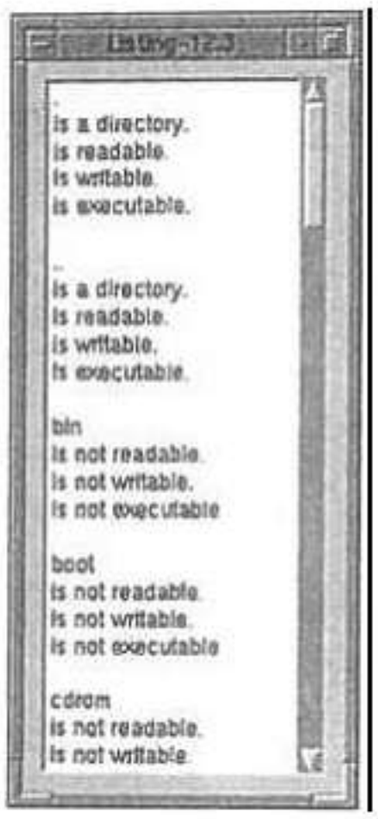


图 12-3　程序清单 12-3 列出目录调查程序，正如你看到的，前两个项目为(.)和(..)

在第 22、23 和 24 行，创建一个 QMultiLineEdit 对象。这个对象用于显示所查找到的文件。在第 26 行，为/目录（当程序在 Windows 平台下编译时，它将被转换为 C:\）创建 QDir 对象。之后，第 37 行启动一个 for 循环。在这个循环里，QFileInfo 对象（finfo）指向当前项目。在此之后，使用几个 if 和 else 语句检测项目属性。为此，用到 QFileInfo 成员函数。QMultiLineEdit 对象中所写入的内容决定于这些函数的返回结果。

这个程序的功能是检索/目录中的每一个项目。目录中的每一个项目被 for 循环处理，几个 if 语句检测当前项目具有哪些属性。图 12-3 为该程序运行结果。

表 12-3 列出 QFileInfo 类中其他一些函数。但是，应该注意，这个表中的一些函数不能用在非 UNIX/Linux 平台下。

表 12-3　　QFileInfo 成员函数

函　数	描　述
QFileInfo::readLink()	返回一个符号连接所指的项目名称
QFileInfo::owner()	返回项目所有者
QFileInfo::ownerID()	返回项目所有者标识
QFileInfo::group()	返回项目所属组名
QFileInfo::groupID()	返回组标识
QFileInfo::permission()	使用这个函数，能够测试一个项目是否具有指定属性。例如，QFileInfo::permission(QFileInfo::WriteUser)将测试项目是否能够被所有者（用户）写入。如果能够，QFileInfo::permission()将返回 TRUE（更多信息请参看 Qt Reference Document）
QFileInfo::size()	返回项目长度（多指文件）
QFileInfo::lastModified()	返回一个 QDateTime 对象，说明项目最后修改日期和时间
QFileInfo::lastRead()	返回一个 QDateTime 对象，说明项目最后被读取的日期和时间

QFileInfo 是一个伟大的发明：它能够读取一个文件的所有信息，像文件大小和它的权限、什么时候被读取及其所有者等等。更重要的是，与其他所有 Qt 类一样，它也非常容易使用。

12.4　小　结

正如这一章前面部分所提到的，当使用其他库开发程序时，为了处理文件和目录，我常选用 Qt 类。我甚至在一些文本程序中也使用这些类。尽管你可能对我的编程习惯不感兴趣，但它们只是向你说明这些类具有多么强大的功能。

你或许使用过标准 C++类（或者甚至是 C 函数和结构）处理文件系统，并且对它们感到非常习惯。或许根本就不计划在自己所用平台以外的其他平台下使用你的程序。如果是这样的话，可以继续使用你熟悉的方法。但是，有一点应该记住：在处理文件和目录时，Qt 的功能非常强大。一旦学会使用它们（这也不会花费很多时间），你将再不想使用其他工具。

12.5　问题与答案

问：我使用 Windows，不理解所谈论的用户、组、文件权限等等。这些内容是什么？

答：这些不适用于 Windows 平台，而只适用于 UNIX。不使用 UNIX 特有函数，你的程序照样能够正常工作。

问：我不能打开想要处理的文件，这是什么原因？

答：要确保你具有文件打开权限。这个问题最常见的原因是权限不够。

问：在程序清单 12-3 中，在 for 循环的尾部，我看到这行：

```
medit->insertLine( " " ) ;
```

为什么使用这行？将行号增加 2 而不是 1 不是更简单吗？

答：实际上，在这种情况下你不能那样做。QMultiLineEdit::insertLine()函数的工作方式是这样：如果传递给它的行号大于当前行号，文本将被添加到最后一行之后的那一行上。因此，这里添加一个空行，以使项目描述之间保留一定的空间。

12.6　作　业

完成下面的问题和练习有助于你理解 Qt 中文件和目录的处理方法。

12.6.1　测验

1. 在 Qt 中哪个类代表一个文件？
2. QTextStream 类的用途是什么？
3. QDir 类做什么用？
4. 怎样读取一个目录中的第 5 个项目？
5. 什么时候需要使用 QFileInfo 类？

12.6.2　练习

1. 使用 QDir 遍历一个目录。然后用 QFileInfo 读取目录中每个文件长度。最后，在屏幕上打印出每个文件名及其长度（选择一个好的 Qt 部件来显示该信息）。
2. 编写一个简单的程序，它使用 Qt 类将一些文本保存到文件中。如果需要，你可以使用一个 QMultiLlineEdit 对象让用户写入文本，之后将这些文本保存到文件里。注意，你需要查阅 Qt Reference Document，因为本课中没有介绍文件的保存过程。
3. 编写一个简单的程序，它使用 QDir 类创建一个目录，之后在那个目录中创建一些文件。
4. 编写一个程序，用它删除练习 3 程序中所创建的文件和目录。当然，应该使用 Qt 类来实现。

第 13 学时　处理文本和理解常规表达式

如果需要从用户读取某类信息，如用户名称或电子邮件地址等，就需要输入验证功能。这一功能应确保用户输入正确的信息。例如，当需要用户输入电子邮件地址时，就不希望输入其名称。

程序中需要验证被提交的文本，以确保它就是程序所需信息。为此，Qt 提供了 QValidator 类。它是一个抽象类，你可以基于它创建新类，并准确定义什么是有效信息和什么是无效信息。验证类能够与 QLineEdit、QSpinBox 和 QcomboBox 一起使用。

为了是有效信息定义变得更加简单，Qt 在 QRegExp 类中提供大量常规表达式定义。它们提供一种定义未知字符或未知字符数字的方法。使用常规表达式使创建新的验证类过程变得更加简单。

在这一课里，将学习怎样使用 QValidator 和 QRegExp 来创建验证函数。

13.1　常规表达式

这课一开始我们先来学习 QRegExp 所包含的常规表达式。这很自然，因为验证类的创建或多或少依赖于常规表达式。如果你在其他一些编程语言中使用过常规表达式，例如 Perl，对这一节的内容就不会感到陌生。但是，对于不熟悉常规表达式的人来说，这一节将很有用处。

如前所述，一个常规表达式是一个字符，它表达一个或多个其他字符。你很可能已经看到常规表达式*，它表示任意字符。其常用方法是定义文件类型。例如，*.txt 表示以.txt（文本文件）结尾的所有文件。使用 Qt 所带的所有不同常规表达式，能够很容易地创建验证类，由它定义哪些是你感兴趣的字符串以及哪些不是。

实际上，在 Qt 中有两类不同的常规表达式。第 1 类为元字符，它表示一个或多个常量字符（依赖于你所指的是哪个元字符）。另外一类为转义序列。与元字符相反，转义字符代表一个特殊字符。

13.1.1　元字符

先来讨论元字符。你将发现这类常规表达式最有用。所有元字符及其解释如表 13-1 所示。

表 13-1　　元字符

常规表达式	解　　释
.	匹配任意单个字符。例如，1.3 可能是 1，后跟任意字符，再跟 3
^	匹配字符串首字符。例如，^12 可为 123，但不能为 312
$	匹配字符串结尾字符串。例如，12$可为 312，但不能为 123
[]	匹配你在括号内输入的任意字符。例如，[123]可以为 1、2 或 3
*	匹配任意数量的前导字符。例如，1*2 可以为任意数量个 1（甚至没有一个），后跟一个 2
+	匹配至少一个前导字符。例如，1+2 必须为一个或多个 1，后跟一个 2
?	匹配一个前导字符或为空。例如，1?2 可以为 1 或 1 2

通过调用 QRegExp::setWildcard(TRUE)可以将元字符设置为统配模式。在统配模式下，只有 3 个元字符可以使用，它们的功能也有一点改变，这些元字符如表 13-2 所示。

表 13-2　　统配模式下的元字符

常规表达式	解释
?	匹配任意单个字符。例如，1?2 可以为 1，后跟任意单个字符，再跟 2
*	匹配任意一个字符序列。例如，1*2 可以为 1，后跟任意数量的字符，再跟一个 2
[]	匹配一个定义的字符集合。例如，[a-zA-Z\.]可以匹配 a 到 z 之间的任意一个字符（大写或小写）和点。[^a]匹配除小写 a 以外的所有字符

如果在 Linux/UNIX 提示符下使用过常规表达式，你可能对统配模式下的元字符更加熟悉。选择使用哪种模式完全由个人爱好和所要实现的操作决定。

13.1.2　转义序列

另一种常规表达式称做转义序列。这些常规表达式用于表达那些不能常规输入的字符，如*和?。这些常规表达式如表 13-3 所示。

表 13-3　　转义序列

常规表达式	解　　释
\.	匹配 "."
\^	匹配 "^"
\$	匹配 "$"
\(	匹配 "("
\)	匹配 ")"
*	匹配 "*"
\+	匹配 "+"
\?	匹配 "?"
\b	匹配响铃字符，使你计算机发出嘟一声
\t	代表制表符
\n	换行符

续表

常规表达式	解　　释
\r	回车符
\s	表示任意空格
\xnn	匹配十六进制值为 nn 的字符
\0nn	匹配八进制值为 nn 的字符

正如你所看到的，所有这些表达式均以一个反斜线（\）开头。C++也是用这种方法表示某些字符。例如，\字符在 C++中应被输入为\\。因此，为了使用 QRegExp 中定义的一个转义序列，你需要再添加一个\，以告诉程序你实际想打印“\”。因此，为了匹配“.”，实际需要输入为“\\.”。

13.2　预定义验证类

在这一节，将学习怎样使用预定义验证类 QDoubleValidator 和 QIntValidator。使用这两个类能够验证双精度值和整数值。你将学习两个例子来了解怎样实现这一点。

13.2.1　QDoubleValidator 类

QDoubleValidator 类用于校验浮点数。例如，如果你需要用户输入 1 至 2 之间的浮点数，则可以使用 QDoubleValidator 类以确保数字真正在 1 和 2 之间，而不是其他值。QDoubleValidator 类非常容易使用，程序清单 13-1 就是一个例子。

程序清单 13-1　　QDoubleValidator 类使用例子

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qlineedit.h>
 4: #include <qvalidator.h>
 5: #include <qmessagebox.h>
 6:
 7: class MyMainWindow : public QWidget
 8: {
 9:     Q_OBJECT
10: public:
11:     MyMainWindow();
12: private:
13:     QDoubleValidator *dvalid;
14:     QLineEdit *edit;
15: public slots:
16:     void slotReturnPressed();
17: };
18:
19: //Define the slot which will be called
20: //when the user has entered a valid string
21: //and then presses Enter:
22: void MyMainWindow::slotReturnPressed()
```

```
23: {
24:     QMessageBox *mbox = new QMessageBox( "Validation Results",
25:                 'If you see this message, the string
26:                 you entered in the QLineEdit object is
27:                 acceptable.", QMessageBox::Information,
28:                 QMessageBox::Ok, 0, 0 );
29:     mbox->show();
30: }
31:
32: MyMainWindow::MyMainWindow()
33: {
34:     setGeometry( 100, 100, 200, 50 );
35:
36:     //We make the validator accept values from 0
37:     //to 10, with one decimal:
38:     dvalid = new QDoubleValidator( 0.0, 10.0, 1, this );
39:
40:     edit = new QLineEdit( this );
41:     edit->setGeometry( 10, 10, 180, 30 );
42:     edit->setValidator( dvalid );
43:
44:     //We connect the signal returnPressed() of
45:     //our QLineEdit object to our custom slot
46:     //slotReturnPressed()
47:     connect( edit, SIGNAL( returnPressed() ),
48:         this, SLOT( slotReturnPressed() ) );
49: }
50:
51: void main( int argc, char **argv )
52: {
53:     QApplication a(argc, argv);
54:     MyMainWindow w;
55:     a.setMainWidget( &w );
56:     w.show();
57:     a.exec();
58: }
```

你可能已经理解程序清单中的大部分内容。但是，有几项新内容需要进一步解释。

首先，需要了解传递给 QDoubleValidator 类的参数代表什么意思。这是一个非常简单的概念：第 1 个参数表示验证者能够接受的最小值，第 2 个参数表示最大值，第 3 个参数表示验证者能够接受的最多小数位数。在这里，构造函数将接受 0.0 到 10.0 之间最多只有一个小数位的所有值。

之后，告诉 QLineEdit 对象（edit）使用 dvalid 作为其验证者。这由下面一行代码实现：

```
edit->setValidator( dvalid ) ;
```

这一过程与处理 QComboBox 和 QSpinBox 对象完全相同，它们也具有 setValidator()函数。

现在，编译并执行该程序。其执行结果如图 13-1 所示。



图 13-1 一个简单的 QLineEdit 对象，它具有验证者功能

如果你想在行编辑器中输入一些文本，你将发现程序不会接受它们，并没有任何字符被显示出来。相反，行编辑器接受数字，因此，它们将被显示在屏幕上。但是，这里所定义的验证者只接受 0.0 和 10.0 之间的数字（最多带一位小数）。你可以输入一个此范围外的数字然后按回车键来测试这一点。例如，试试 123。因为该数字验证者不能接受，所以不会发生任何事。但是，如果你输入一个有效数字，例如 4.3，slotReturnPressed()槽将被调用，屏幕上将显示出图 13-2 所示的窗口。

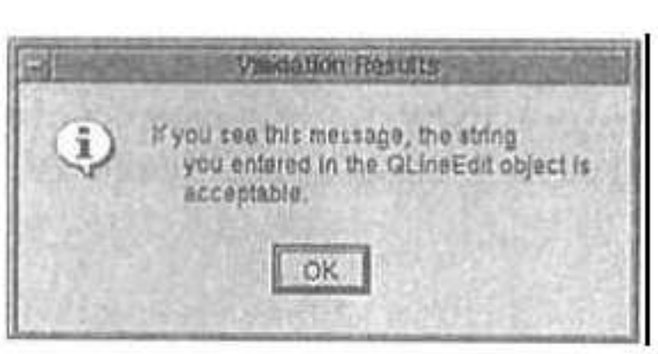


图 13-2　输入一个有效数字并按回车之后所显示的消息框

如果你想测试更多数字，只需点击 OK 按钮，之后再输入其他数字。这个简单的例子阐明了验证者这一概念：需要的数字（或字符）被接受，而不需要的数字（或字符）被忽略。这能够确保得到你想要的数字，而不必担心超出范围的数字。

13.2.2　QIntValidator 类

Qt 所带的另一个预定义验证者为 QIntValidator 类。前面已经提到，这个类用于验证整数，与 QDoubleValidator 类相反，它用于验证浮点数。你可以像使用 QDoubleValidator 类一样使用 QIntValidator 类。唯一区别是省略代表最多小数位数的构造函数参数。毕竟 QIntValidator 类只用于整数。

你可以修改程序清单 13-1 来使用 QIntValidator 对象，只需要将所有的 QDoubleValidator 实例修改为 QIntValidator。之后，只要删除 QIntValidator 构造函数的第 3 个参数，并将第 1 和第 2 个参数中的 double 值修改为 int 值。这样做将把程序清单 13-1 转换为一个整数验证者而不是浮点数。

13.3　创建用户验证类

预定义验证类能够验证浮点数和整数，但你很可能需要验证其他内容（如字符或字符和数字的混合）。例如，需要验证一个电子邮件地址。这正是在这一节里你将要学习到的内容。

使用 QValidator 作为基类能够创建用户验证类。在程序清单 13-2 中，你将学习怎样创建一个类用于验证电子邮件地址。使用你新学到的常规表达式知识，完成这一任务并不困难。

程序清单 13-2　　电子邮件验证类

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qstring.h>
4: #include <qlineedit.h>
5: #include <qvalidator.h>
```

```
 6: #include <qmessagebox.h>
 7: #include <qregexp.h>
 8:
 9: // ----- Start definition of custom validator -----
10: class EMailValidator : public QValidator
11: {
12: public:
13:     EMailValidator();
14:     QValidator::State validate( QString&, int& ) const;
15: };
16:
17: //We don't need to do anything in the constructor:
18: EMailValidator::EMailValidator() : QValidator(0)
19: {
20: }
21:
22: QValidator::State EMailValidator::validate( QString &text, int &pos ) const
23: {
24:     //Create a regular expression that matches any
25:     //e-mail address with a top-domain of three characters:
26:     QRegExp regexp( "*@*.???", FALSE, TRUE );
27:
28:     //Store the result of the query in result:
29:     int result = regexp.match( text );
30:
31:     //If the string matches the query, return Acceptable:
32:     if( result != -1 )
33:     {
34:         return QValidator::Acceptable;
35:     }
36:     //If it doesn't match, return Invalid:
37:     else
38:     {
39:         return QValidator::Invalid;
40:     }
41: }
42: //----- End definition of custom validator -----
43:
44: //----- Start Definition of MyMainWindow -----
45:
46: //Remember to make a .moc file of this class
47: //declaration and then include that file instead:
48: class MyMainWindow : public QWidget
49: {
50:     Q_OBJECT
51: public:
52:     MyMainWindow();
53: private:
54:     EMailValidator *evalid;
55:     QLineEdit *edit;
56: public slots:
57:     void slotReturnPressed();
58: };
59:
60: void MyMainWindow::slotReturnPressed()
61: {
62:     QMessageBox *mbox = new QMessageBox( "Validation Results",
63:                 "If you see this message, the string
64:                 you entered in the QLineEdit object is
```

```
65:                 acceptable.", QMessageBox::Information,
66:                 QMessageBox::Ok, 0, 0 );
67:     mbox->show();
68: }
69:
70: MyMainWindow::MyMainWindow()
71: {
72:     setGeometry( 100, 100, 200, 50 );
73:
74:     evalid = new EMailValidator();
75:
76:     edit = new QLineEdit( this );
77:     edit->setGeometry( 10, 10, 180, 30 );
78:     edit->setValidator( evalid );
79:
80:     connect( edit, SIGNAL( returnPressed() ),
81:          this, SLOT( slotReturnPressed() ) );
82: }
83: //----- End definition of MyMainWindow -----
84:
85: void main( int argc, char **argv )
86: {
87:     QApplication a(argc, argv);
88:     MyMainWindow w;
89:     a.setMainWidget( &w );
90:     w.show();
91:     a.exec();
92: }
```

当在前一个例子中使用 QDoubleValidator 和 QIntValidator 时，预定义 validate()函数用于判断所输入的字符串是否能够接受。但是，这里你需要定义自己的 validate()函数版本（第 22 行到第 41 行），因为它是 QValidator 的一个虚函数。在 EmailValidator::validate()中，使用 QRegExp 创建一个常规表达式（第 26 行），然后使用 QRegExp::match()检查所输入的字符串是否与这个表达式相匹配。如果匹配，用返回值 QValidator::Acceptable 告诉验证者接受这个字符串（第 32 到 35 行）。如果不匹配，则返回 QValidator::Invalid（第 37 到 40 行）。

之后，在 MyMainWindow 类中使用这个新的验证类。这里，创建一个 QLineEdit 对象（第 76、77 和 78 行），并使用 EmailValidator 类验证该对象中的输入（第 78 行）。也创建一个槽 slotReturnPressed()（第 60 到 68 行），这个槽使用 QMessageBox 向用户显示信息。slotReturnPressed()被连接到 QLineEdit::returnPressed()信号（第 80 和 81 行），当用户按回车键时发射该信号。

这样做的结果是当你所输入的电子邮件地址与<任意字符>@<任意字符>.<3 个字符>相匹配时，将显示一个消息框。如果你输入其他内容时，则不会发生任何事。

你可能已经注意到，这个验证者不能接受所有的电子邮件地址。这是因为顶级域被定义为只有 3 个字符组成。如果你电子邮件的顶级域只有两个字符，该地址将不被接受。但是，这很容易修正。

这个例子中所用到的概念可用于创建所有的用户验证类。通过使用 QRegExp 所提供的功能强大的常规表达式，编写验证类工作将十分简单。

fixup()函数

如果在 Reference Document 中查阅 QValidator，你可能注意到这个类还有另一个虚函数，fixup()。这个函数用于纠正不可接受但又不是无效的字符串。实际上，QValidator 为这一状

态定义一个特殊类型：QValidator::valid。如果你发现所输入的字符串不可接受，但又可能使它们变为可接受的字符串时，应该返回 QValidator::valid，之后 QValidator 将调用 fixup()函数，该函数中应该包含必要的代码使字符串转换为可接受的。

例如，如果你只想接受.org 顶级域下的电子邮件地址，应该在 validate()中进行检查：

```
if ( text[pos - 3] != 'o' || text[pos - 2] != 'r' || text[pos - 1] != 'g' )
{
    return QValidator::valid
}
```

这里，如果顶级域不是.org，则将返回 QValidator::valid，并调用 fixup()函数。下面这个例子说明在这种情况下怎样使用 fixup()函数：

```
void EmailValidator::fixup( QString &text )
{
    int pos = text.length( ) ;
    text[pos - 3] = 'o'  ;
    text[pos - 2] = 'r'  ;
    text[pos - 1] = 'g'  ;
}
```

这段代码确保地址以 o、r 和 g 三个字符结尾。这个函数有助于纠正用户所产生的小错误，或者将所输入的字符串转换为正确格式。例如，如果你想让用户输入其名字和电话号码，并先输入名字。这时，用户很可能会误解这一点，而先输入电话号码。使用 fixup()函数就很容易纠正这一点。

13.4　小　结

无论什么时候，只要你觉得用户可能输入错误信息时，就应该使用验证类。

可以用不同的方式实现实际验证操作（在 validate()中的代码）。例如，可以选择使用统配模式。如前所述，这只是一个爱好和需要问题。

最后提醒一点，你不要使验证过于严格。换句话说，最好使基本能够接受的字符串都通过。

13.5　问题与答案

问：我大量使用 MS-DOS，因此，我认为在统配模式下使用常规表达式最适合我。但是，它们与我过去所使用的不完全相同。是这样吗？

答：是的，正如你在表 13-2 中所看到的，统配模式下的元字符效果与 UNIX 下的常规表达式相同。因此，它们不完全与 MS-DOS 下的相同。

问：我无法使用转义序列，这是什么原因？

答：应该记住在 C++下需要输入两个反斜杠线才能得到一个反斜杠。因此，在使用

QRegExp 中定义的转义序列时，你也需要使用两个反斜杠。

问：转义序列\b 是什么？它有什么用途？

答：测试下面一行代码：

```
count << "\b" ;
```

如果你的计算机能够发声（它具有所谓的 PC 扬声器），当运行这一代码时，你将听到发自计算机的嘟声。

13.6　作　业

学习这一节，其中的这些问题和练习将有助于你牢记刚学过的文本处理和常规表达式知识。

13.6.1　测验

1. 为什么你认为 QValidator 是一个抽象类？
2. 是否有预定义验证类？
3. 如果你在前一个问题中回答是，那么你能告诉这一验证类（或这些验证类）的用途是什么吗？
4. 哪个 Qt 类能够使用验证类？
5. 你需要调用哪个函数来告诉一个对象使用验证？
6. 当创建你自己的验证类时，实际验证代码插入在哪个虚函数内？
7. 你可用某个虚函数纠正基本可接受的字符串，这个函数是什么？

13.6.2　练习

1. 重新编写程序清单 13-2，使它也能够接受由两个字符组成的顶级域地址。
2. 使用 QIntValidator 创建一个校验温度的验证类。使它只接受你那里的合理温度值。
3. 为国际标准书号（ISBN）编写一个验证类。这些号码使用 X-XXXXX-XXXX 格式（实际上 ISBN 格式可以有一些变化，为了简化，只使用这里指定的格式）。验证类应该保证所输入的字符串格式正确。

第 14 学时　学习使用容器类

在这一学时，将学习 Qt 的另一种技术：容器类（Container class）的用法。你可以使用这些类存储其他元素（如其他类对象）。Qt 一共带有几个容器类，它们构成一个完整的容器类库，这足以完成所有工作。

在开始之前，你应该意识到：如果不熟悉 C++内存管理技术的话，这一学时中的一些内容可能很难理解。但是，不要紧张，每一个问题都将逐步得到解决。

14.1　Qt 容器类

如前所述，Qt 容器类用于存储其他元素（常为其他类对象）。但是，有几种不同方法可以用来存储和访问容器类中的对象。因此，Qt 提供几个不同的容器类，尽管其中一些工作非常类似，但它们均用于不同需要。

Qt 容器类总体情况如表 14-1 所示。这个表说明 Qt 容器类的不同用法。阅读这个表可以总体了解不同容器类的用法。

表 14-1　　Qt 容器类

类	描　述
QArray	这是一个非常简单的类，只能存储非常简单的元素。它不能存储内置对象以外的其他任何对象，并且要求其元素不能有构造函数、析构函数和虚函数
QBitArray	这是从 QArray 派生出的一个类。然而，它只能存储位。当要搜集程序中的标识信息时，它非常有用
QPointArray	这是又一个从 QArray 派生出的类。它用于存储 QPoint 对象
QDict	这个类将其元素存储在散列（hash）表中。散列表中的元素是通过字符串来访问。例如，你可以将一些字符串对存储在 QDict 对象中，然后通过一个字符串来访问另一个字符串
QIntDict	与 QDict 类工作方式相似，但它使用 long 值而不是 char*
QPtrDict	与 QDict 相同，但它使用数据而不是 char*做键（代表散列表中元素的字符串）
QCache	QCache 工作与 QDict 相同。但是，你不能控制 QCache 中的元素数量，因此，列表不能太长
QIntCache	与 QCache 相同，但它使用 long 值而不是 char*
QList	这个类提供一个双向来链接列表，在 C++程序设计中它是一个普通的元素。这和列表功能很强，它使你能够使用所有方法插入和访问其中的元素。更多信息请参看 Qt Reference Document
QStrList	这个类派生于 QList，它用于存储字符串列表
QStrIList	与 QStrList 类相同，但它在执行比较操作时区分字母大小写
QStack	当需要创建一个列表，并用反序（后进先出）访问其中的元素时应使用这个类
QQueue	当你想创建一个列表，并用与插入相同的顺序（先进先出）访问其中的元素时应该使用这个类

这些类构成一个完整的容器集合，它们可以存储所有种类的信息，从单个位到完整的类对象。最小类，QBitArray，只能存储位（1 和 0），相反，QList 类提供一个所有 Qt 类对象的双向链接列表。

通过学习使用这些类，你能够使用所有种类的元素创建任意类型的列表。在下一节中将更详细地讨论这些类的用法

14.2　栈和队列

栈是程序员熟知的一种数据结构。当需要向列表中插入元素（所有种类），之后用相反顺序（后进先出）检索它们时使用栈。

队列与栈相反。使用这种列表，你可以用与插入时相同的顺序（先进先出）检索元素。

在 Qt 中，用 QStack 和 QQueue 分别创建栈和队列。这些类的使用与其他类有一点不同，因此，下面将单独讨论这两种类。首先从怎样使用 QStack 类开始。

14.2.1　用 QStack 类创建栈

如前所述，当需要创建一个对象列表，并用相反顺序检索它们时应该使用栈。栈在很多程序设计环境中是非常有用的。例如，假设你正在创建一个用户能够添加图标的 Qt 部件但是，你想使这个部件一次所具有的图标数不能超过 10 个。如果当部件满时再添加图标，最后插入的一个应当被删除，以便为新插入的图标提供空间。使用栈就很容易实现该操作。

程序清单 14-1 给出一个例子，它说明怎样使用 QStack。这里，创建一个能够存储字符串（字符）的栈。3 个元素被插入到栈中。最后检索元素它们（并同时删除它们），并在一个 QMultiLineEdit 对象中显示出来。

程序清单 14-1　　用 QStack 创建栈

```
 1: #include <qapplication.h>
 2: #include <qapplication.h>
 3: #include <qwidget.h>
 4: #include <qmultilineedit.h>
 5: #include <qstack.h>
 6:
 7: void main( int argc, char **argv )
 8: {
 9:     QApplication a(argc, argv);
10:     QWidget w;
11:     w.resize( 150, 150 );
12:     a.setMainWidget( &w );
13:
14:     QMultiLineEdit edit( &w );
15:     edit.setGeometry( 10, 10, 130, 130 );
16:
17:     //Create a stack for the char elements:
18:     typedef QStack<char> StringStack;
19:     //Create an object of this stack:
20:     StringStack stringstack;
21:     //Insert a few elements to the stack:
```

```
22:     stringstack.push( "Element 1" );
23:     stringstack.push( "Element 2" );
24:     stringstack.push( "Element 3" );
25:
26:     //While there's any element left in the stack,
27:     //get it and output it the QMultiLineEdit
28:     //object. Note that the pop() function also
29:     //removes one object from the stack.
30:     while( stringstack.current() )
31:     {
32:         edit.insertLine( stringstack.pop() );
33:     }
34:
35:     w.show();
36:     a.exec();
37: }
```

这个例子中的所有内容都很直观。但是，第 18 行内容看起来可能比较陌生：

```
typedef QStack<char> StringStack ;
```

这一行创建一个处理 char 元素的特殊类型栈，这个栈叫做 StringStack。如果你想创建一个 int 元素栈，这一行将改为：

```
typedef QStack<int> IntStack ;
```

正如你所看到的，你只需改变两个尖括号间的类型。栈名也可以改变，但这当然不是必须的——你喜欢怎样叫它就可以怎样叫它。



你很可能在此之前还没有看到尖括号的这种用法。但是，它并不神秘，这只是一种定义容器类存储类型的 C++方法。然而，如果你使用过 STL（Standard Template Library、标准模板库），对此你就不会感到陌生。



注意，缺省时栈用指针处理其元素。所以，不必这样输入：

```
typedef QStack<char*> StringStack
```

栈总是存储指针。如果你使用这一行，将变为存储指向指针的指针。这适用于除 QArray 和 QBitArray 以外的其他所有容器类。

与其他类一样，可以用下面一行创建该类对象：

```
StringStack stringstack;
```

然后，即可使用该类成员函数向栈中插入元素和从栈中检索元素。QStack::push()用于插入元素；QStack::pop()用于检索元素（QStack::pop()也用于删除它从列表中所检索的元素）；QStackCurrent()用于检索元素但不删除它。为了从栈中删除一个元素而不检索它，应使用 QStack::remove()函数。程序清单 14-1 所创建的程序如图 14-1 所示。

正如你所看到的，元素以相反的顺序被检索出来，即先检索出 Element 3。如前面所讨论的，这是因为 QStack 从元素插入的一端检索元素。如果使用队列（QQueue），将以相反的顺序将元素插入到 QMultiLineEdit 中，从 Element 1 开始。



图 14-1　一个 QMultiLineEdit 对象，它具有从栈中检索出 3 个元素

14.2.2　用 QQueue 类创建队列

与栈相反，队列从一端插入元素，并从另一端检索元素。使用这种方法，你能够以插入时相同的顺序从队列中检索元素。

因此，如果你想修改上一节中的图标部件例子，使它在为新的图标提供空间时是删除第 1 个元素而不是最后 1 个元素，只需使用 QQueue 代替 QStack 即可。这将使那个列表用做一个真正的队列：先进先出。

程序清单 14-2 很类似于程序清单 14-1，它只是使用 QQueue 代替 QStack。这一结果将使元素以正确的顺序显示（与它们的插入顺序相同）。

程序清单 14-2　　用 QQueue 创建队列

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qmultilineedit.h>
4: #include <qqueue.h>
5:
6: void main( int argc, char **argv )
7: {
8:     QApplication a(argc, argv);
9:     QWidget w;
10:    w.resize( 150, 150 );
11:    a.setMainWidget( &w );
12:
13:    QMultiLineEdit edit( &w );
14:    edit.setGeometry( 10, 10, 130, 130 );
15:
16:    typedef QQueue<char> StringQueue;
17:    StringQueue stringqueue;
18:    stringqueue.enqueue( "Element 1" );
19:    stringqueue.enqueue( "Element 2" );
20:    stringqueue.enqueue( "Element 3" );
21:
22:    while( stringqueue.current() )
23:    {
24:        edit.insertLine( stringqueue.dequeue() );
25:    }
26:
27:    w.show();
28:    a.exec();
29: }
```

正如你所看到的，QQueue 很像 QStack。但是 QQueue 使用 enqueue()和 dequeue()来插

入和删除元素而不是 push()和 pop()。图 14-2 显示出程序清单 14-2 所创建的程序运行结果。



图 14-2　一个 QMultiLineEdit 对象，它具有从队列中检索出的 3 个元素

前面已经讲过，因为这里使用队列，所以检索出的元素顺序与它们的插入顺序相同，即先为 Element 1。

没有任何规则能够告诉你什么时候应该使用队列而不是栈。这总体上依赖于程序的组织方式和你的当前需要。但是，如前所述，当需要以插入时相同的顺序检索元素时就应该使用队列，而当需要以相反的顺序检索元素时，则应该使用栈。注意，元素可以为任意内容，从单个位到 Qt 部件。

14.3　散列表

散列表是一个列表，它能够通过字符串键值检索对象。这在很多情况下非常有用。例如，如果你在创建一个电子邮件程序，并实现通信列表，就可以很容易地创建一个名称和电子邮件地址散列表。之后，可以通过指定名称来检索电子邮件地址。或者反过来，通过指定电子邮件地址来检索名称。程序清单 14-3 所给出的例子说明怎样使用散列表。

程序清单 14-3　　用 QDict 创建散列表

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qmultilineedit.h>
 4: #include <qdict.h>
 5:
 6: void main( int argc, char **argv )
 7: {
 8:     QApplication a(argc, argv);
 9:     QWidget w;
10:     w.resize( 150, 150 );
11:     a.setMainWidget( &w );
12:
13:     QMultiLineEdit edit( &w );
14:     edit.setGeometry( 10, 10, 130, 130 );
15:
16:     typedef QDict<char> StringDict;
17:     StringDict stringdict;
18:     stringdict.insert( "Sweden", "Stockholm" );
19:     stringdict.insert( "Germany", "Berlin" );
20:     stringdict.insert( "France", "Paris" );
21:     stringdict.insert( "England", "London" );
22:
23:     edit.insertLine( stringdict["England"] );
```

```
24:     edit.insertLine( stringdict["France"] );
25:     edit.insertLine( stringdict["Germany"] );
26:     edit.insertLine( stringdict["Sweden"] );
27:
28:     w.show();
29:     a.exec();
30: }
```

这个例子非常简单。首先，在第 16、17 行创建一个字符串散列表。之后在第 18 到 21 行向该散列表中插入 4 个元素。最后，在第 23 行到第 26 行检索这些元素并将它们插入到 QMultiLineEdit 对象中。注意，[]操作符用于访问字符串。图 14-3 为该程序运行结果。



图 14-3　一个 QMultiLineEdit 对象，它具有从散列表中检索出 4 个字符串



注意，传递给 insert()的第 2 个参数为实际元素。第 1 个参数为字符串键。因为在这个例子中字符串被用做元素，所以它可能有点难以理解。



不要忘了另一个目录功能：QIntDict。它用于存储整数值。能使用 QDict 的地方，也可以使用 QIntDict。

如你在程序清单 14-3 中所看到的，字符串必须被逐个打印出来——不能使用 while 循环之类的语句。相反，你必须使用迭代来遍历一个散列表。本章后面部分将讨论迭代。

QDict 还有几个可能对你有用的函数，像 QDict::clear()（清除整个散列表）和 QDict::isEmpty()（检查散列表是否为空）。欲了解更多这方面内容请参考 Qt Reference Document。

14.4　数据缓存

QCache 和 QIntCache 能够用于创建具有大小受限的散列表。这意味着你可以设置容器最多能够变到多大，如果达到这个大小，使用最少的元素将被删除。如果想得到准确数量的信息，或是想控制一个散列表所具有的元素数以减少内存消耗时，这非常有用。程序清单 14-4 给出一个例子。

程序清单 14-4　　用 QCache 创建大小受限散列表

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qmultilineedit.h>
4: #include <qcache.h>
5:
6: void main( int argc, char **argv )
7: {
8:     QApplication a(argc, argv);
9:     QWidget w;
10:    w.resize( 150, 150 );
11:    a.setMainWidget( &w );
12:
13:    QMultiLineEdit edit( &w );
14:    edit.setGeometry( 10, 10, 130, 130 );
15:
16:    typedef QCache<char> StringCache;
17:    StringCache stringcache;
18:    //We set the maximum total cost to 6:
19:    stringcache.setMaxCost( 6 );
20:    //We let the cost for each element be 2
21:    //(default is 1):
22:    stringcache.insert( "Sweden", "Stockholm", 2 );
23:    stringcache.insert( "Germany", "Berlin", 2 );
24:    stringcache.insert( "France", "Paris", 2 );
25:
26:    //Here, the total cost has already reached 6,
27:    //so what will happen?
28:    stringcache.insert( "England", "London", 2 );
29:
30:    edit.insertLine( stringcache["England"] );
31:    edit.insertLine( stringcache["France"] );
32:    edit.insertLine( stringcache["Germany"] );
33:    edit.insertLine( stringcache["Sweden"] );
34:
35:    w.show();
36:    a.exec();
37: }
```

这里，在第 16、17 行创建一个缓存。第 19 行为这个散列表创建最大成本（解释见下）。在第 22 到第 28 行，4 个元素被插入到表中。每个元素的成本被设置为 2。在第 30 到第 33 行，检索元素并将它们插入到 QMultiLineEdit 对象中。该程序运行结果如图 14-4 所示。

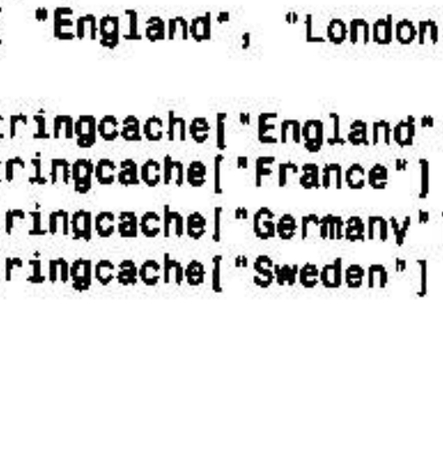


图 14-4　一个 QMultiLineEdit 对象，它具有从 QCache 对象中检索出的 3 个字符串

像你所看到的，从缓存中只得到 3 个字符串。Stockholm 到哪里去了？唯一的解释是在

最后一个元素被添加到缓存之前已经达到最大总成本。因此，缓存简单删除最长时间未被访问的元素。因为，所有元素都未被访问，所以，删除第 1 个插入的元素。

QCache 的用法与 QDict 非常类似。但是，应该注意怎样使用 setMaxCost()函数设置最大总成本和通过向 insert()函数添加第 3 个参数来定义每个元素的成本。如果不为每个元素定义成本，成本默认值为 1。

QIntCache 的使用与 QCache 一样，但它应该用于存储整数而不是字符串。但是，使用一个数来引用另一个数有用吗？事实上，它有用。如果你有几个整数需要以某种方式组织，你可以将它们添加到一个 QIntcache 中并对它们编号，如 1、2、3 等等。使用这种方式，能够很容易地通过引用序号来检索整数。

14.5 迭 代

为了遍历一个容器类，最好使用迭代。迭代是一种特殊类，它有特殊用途：遍历容器。尽管这完全可以使用 while 循环之类的语句（像程序清单 14-1 和 14-2 中那样）来实现，但最好还是使用迭代（因为这样遍历更安全）。例如，迭代能够用于在遍历过程中变化的容器。此外，也可以使用迭代来遍历目录类。

每种容器类均有一种类型的迭代：QDictIterator、QCacheIterator、QIntCacheIterator、QIntIterator、QListIterator 和 QPtrIterator。

所有迭代均包含几个用于遍历容器的函数。toFirst()用于提取容器中的第 1 个元素，toLast()用于提取容器中的最后一个元素。迭代可以与++和--操作符一起使用来在容器中前移或后移。current()用于提取当前元素。

程序清单 14-5 对程序清单 14-3 做一些修改，以便它使用迭代来遍历所有元素。

程序清单 14-5 用迭代遍历 QDict 对象

```
#include <qapplication.h>
#include <qwidget.h>
#include <qmultilineedit.h>
#include <qdict.h>

void main( int argc, char **argv )
{
    QApplication a(argc, argv);
    QWidget w;
    w.resize( 150, 150 );
    a.setMainWidget( &w );

    QMultiLineEdit edit( &w );
    edit.setGeometry( 10, 10, 130, 130 );

    typedef QDict<char> StringDict;
    StringDict stringdict;
    stringdict.insert( "Sweden", "Stockholm" );
    stringdict.insert( "Germany", "Berlin" );
    stringdict.insert( "France", "Paris" );
    stringdict.insert( "England", "London" );
```

```
22:
23:     //Create an iterator for our stringdict object:
24:     typedef QDictIterator<char> StringDictIterator;
25:     StringDictIterator stringdictiterator( stringdict );
26:     //Print out the elements to the QMultiLineEdit object,
27:     //one by one:
28:     for( stringdictiterator.toFirst(); stringdictiterator.current();
29:         ++stringdictiterator )
30:     {
31:         edit.insertLine( stringdictiterator.current() );
32:     }
33:
34:     w.show();
35:     a.exec();
36: }
```

这里，在第 16、17 行创建一个 QDict 容器。第 18 到第 21 行向其中插入几个元素。第 24 和 25 行创建一个迭代。在第 28 到第 32 行，检索元素并将它们插入到 QMultiLineEdit 对象中。这由一个 for 循环和迭代来实现。

像你在图 14-5 中所看到的，这个程序清单的运行结果与程序清单 14-3 相同，但其顺序有一点差别。



图 14-5　一个 QMultiLineEdit 对象，它具有从迭代中检索到的 4 个字符串

当然，可以像使用 QdictIterator 一样使用其他迭代。只要保证对不同的容器使用正确的迭代即可。

通过使用迭代代替循环来遍历一个容器，程序将变得更加安全，并且能够减少内存错误。因为迭代被特殊设计用于处理容器遍历，它们能够纠正在容器内步进过程中所出现的多数问题（例如，在遍历期间向容器中添加元素）。一个简单的 for 或 while 循环对此将变得无所适从，这很可能导致程序退出。因此，你应该使用迭代以避免程序错误和其他问题。

14.6　小　结

这里首先介绍了容器的使用。然而，Qt 容器类使整个概念变得更加简单。更准确地说，Qt 容器类使这个概念看起来更简单，因为其内部函数仍是相同的。如果你曾经试图用手工方式创建 Qt 容器类所提供的功能，如一个连接列表，你对其复杂性将有所理解，因此，应该花一点时间学习怎样使用容器类。

14.7　问题与答案

问：我将++操作符与迭代一起使用，但出现警告错误，报告这个操作符不存在。这是什么原因？

答：要将++操作符添加到迭代对象前面，而不是后面。

问：同一个缓存中的元素能够具有不同的成本吗？

答：可以，只要在使用 insert()插入元素时定义它们所具有的成本即可。

问：我能够将自己所创建的类存储到一个容器类中吗？

答：可以，当然能够。只要在定义容器时（以 typedef 开头的程序行）在两个尖括号（<>）之间输入你的类名即可。

14.8　作　业

下面问题和练习将测试你在这一学时中所学到的知识。

14.8.1　测验

1. 什么时候应该使用 QStack 容器类？
2. 什么时候应该使用 QQueue 容器类？
3. 什么是散列表？
4. 如果想创建具有整数键值而不是字符串键值的散列表，应该使用哪个类？
5. QCache 有什么用途？
6. 当一个缓存到达其最大总成本值时将发生什么问题？
7. 什么时候需要使用迭代？

14.8.2　练习

1. 使用 QQueue 创建一个队列。之后编写一个小循环，数出队列中的元素数，并将该数字打印在某个合适的 Qt 部件内。
2. 用 QIntDict 创建一个散列表。向它添加几个元素，之后使用迭代以相反的顺序打印出所有元素。
3. 用 QCache 创建一个缓存，将其最大总成本设置为 20。向该缓存添加几个元素（它们需要具有不同的成本）。确保在你添加最后一个元素之前总成本为 19。现在，你添加最后一个元素，会发生什么问题？

第 15 学时　深入理解图形

在第 9 学时“创建简单图形”中，我们学习了基本的 Qt 图形功能，看到了怎样使用 QPainter 类绘制简单图形、怎样改变颜色以及怎样打印出你所创建的图形。这一学时将继续这一讨论。

在这一学时里学习几个有趣的图形功能。首先，学习怎样在采用图形交换格式（GIF，Graphics Interchange Format）的程序中使用 QMovie 类显示动画。之后，学习怎样装载和保存图像。

15.1　动　画

Qt 提供一个 QMovie 类，用于在程序中显示动画（电影）。当前，QMovie 只能处理 GIF 图像/动画格式。但是，我们期望着将来能够增加其他格式（如 AVI）。而现在，你必须接受这一限制。

创建一个 QMovie 对象，赋给它一个动画，之后将它显示在屏幕上是非常简单的，这一过程程序清单 15-1 所示。

程序清单 15-1　　QLabel 对象中的动画

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qlabel.h>
 4: #include <qmovie.h>
 5:
 6: class MyMainWindow : public QWidget
 7: {
 8: public:
 9:     MyMainWindow();
10: private:
11:     QLabel *label;
12: };
13:
14: MyMainWindow::MyMainWindow()
15: {
16:     setGeometry( 100, 100, 140, 80 );
17:
18:     //Create a QMovie object and tell it the
```

```
19:     //the animation we want to use can be found
20:     //in the file trolltech.gif. We make this
21:     //object a constant reference, otherwise
22:     //QLabel::setMovie won't accept it.
23:     const QMovie &movie( "trolltech.gif" );
24:
25:     label = new QLabel( this );
26:     label->setGeometry( 10, 10, 120, 60 );
27:     //Tell label to use movie:
28:     label->setMovie( movie );
29: }
30:
31: void main( int argc, char **argv )
32: {
33:     QApplication a(argc, argv);
34:     MyMainWindow w;
35:     a.setMainWidget( &w );
36:     w.show();
37:     a.exec();
38: }
```

在这个例子中使用动画 trolltech.gif，该文件位于 Qt 分发程序的 exampes/movies/目录下。这里，为这个动画（电影）创建一个 QMovie 对象（第 23 行），并用一个 QLabel 对象显示这个动画（第 25、26 和 28 行）。用 QLabel::setMovie()函数将这个动画告诉该标签（第 28 行）。要注意，QMovie 对象是一个常量引用，这是因为 QLabel::setMovie()函数不接收其他任何类型。图 15-1 为该程序运行结果。



图 15-1 一个显示 trolltech.gif 动画窗口



如果在运行程序清单 15-1 时你只看到一个空窗口，很可能是在编译 Qt 时没有包含 GIF 支持（像在第一学时“Qt 简介”中提到的）。在运行配置脚本程序时添加-gif 选项即可解决这一问题。因此，如果你没有包含 GIF 支持，你必须像下面这样重新编译 Qt:

```
cd /usr/local/qt
./configure -gif
make
```

然后，重新编译并再次运行程序清单 15-1。

当像程序清单 15-1 中那样启动一个动画后，它将连续循环（一次又一次）显示直到你关闭那个拥有该动画的窗口为止。但是，有几个函数可用于控制动画。这些函数如表 15-1 所示。

表 15-1　　动画控制函数

函　数	描　述
QMovie::pause()	调用这个函数暂停动画
QMovie::unpause()	调用这个函数再次打开已经暂停的动画
QMovie::step()	在暂停一个动画后，可以调用这个函数使它前进一帧。之后，动画再次暂停。实际上，这个函数存在另一个版本，它带一个 int 参数（它是一个重载函数，因此，其名称也为 QMovie::step()，只是参数列表不同）。这个参数表示动画再次暂停之前所显示的帧数
QMovie::restart()	反转动画，如果它未被暂停，则重新开始显示动画
QMovie::setSpeed()	使用这个函数设置动画速度。将速度（一个百分数）作为参数传递给该函数，其缺省值为 100%



如果你不理解步进和动画运行之间的差别，这里对它们加以解释。当运行动画时，帧一直在改变，对象的移动比较平滑。相反，如果调用 QMovie::step()函数，动画将只改变一帧，然后再次停下（好像它是一个图片）。也可以这样说，运行动画相当于无数次地调用 QMovie::step()函数。

QMovie 也包含几个函数用于检测动画当前状态。这些函数如表 15-2 所示。

表 15-2　　检测动画状态函数

函　数	描　述
QMovie::steps()	当你调用 step()函数将动画前进一定步骤之后，可使用这个函数检测剩余步数。如果动画被暂停，将返回 0
QMovie::paused()	如果动画被暂停，这个函数将返回 TRUE
QMovie::finished()	如果动画已经停止和完成（也就是说，所有帧的所有循环已经完成），该函数返回 TRUE
QMovie::running()	如果动画正在运行它返回 TRUE。注意，如果动画处于单步前进状态，将认为它不是处于“运行”状态
QMovie::speed()	返回当前速度百分比

使用表 15-1 和表 15-2 中的函数能够完全控制程序中的所有 GIF 动画。一两个动画就能够是原本乏味的程序看起来更加生动。一个很好的例子就是在所有 Netscape 窗口的右上角都运行着一个小的 Netscape 动画。



如果你正在查找动画 GIF 文件或者构造 GIF 动画所用工具，这里是一个不错的站点：

```
http://dir.yahoo.com/Arts/Visual_Arts/Animation/Computer_Animation/Animated_GIFs/
```

15.2　装载和保存图像

你可能需要将已经创建的图像保存到文件中。这样做，你必须首先决定使用哪种图像

格式。Qt 支持大部分图像格式——PNG、BMP、XPM、XBM 和所有 PNM 格式。前面已经提到，GIF 格式也被支持。但是，你只能浏览（装载）GIF 图像，而不能存储它们。这是由于受 GIF 专利所限。Qt 也有它自己的与平台无关的图像格式（它只能被 Qt 所处理）。

15.2.1　Qt 图像格式

使用 QPicture 类能够记录用 QPainter 类所执行的操作。可以将 QPainter 所绘制的图像用 Qt 图像格式保存到一个文件中。程序清单 15-2 给出一个例子：

程序清单 15-2　　将 QPainter 操作保存到文件

```
#include <qapplication.h>
#include <qwidget.h>
#include <qlabel.h>
#include <qpainter.h>
#include <qpicture.h>

class MyMainWindow : public QWidget
{
public:
        MyMainWindow();
private:
    void paintEvent( QPaintEvent* );
    QPainter *paint;
    QLabel *label;
    QPicture pic;
};

MyMainWindow::MyMainWindow()
{
        resize( 200, 200 );

    paint = new QPainter();
    //Instead of painting in the window,
    //we choose to output to the QPicture
    //object:
    paint->begin( &pic );
    //We draw a rectangle. Not to the screen,
    //but to the QPicture object:
    paint->drawRect( 20, 20, 160, 160 );
    paint->end();

    //Save what has been drawn to the QPicture
    //object to a file, called file.pic. When
    //you have run this program, a file with this
    //name will be created (in the directory which
    //this program is located in).
    pic.save( "file.pic" );
}

//Of course, we also want to see the image
//on screen. This is taken care of by the
//paintEvent() function:
void MyMainWindow::paintEvent( QPaintEvent* )
```

```
44: {
45:     paint = new QPainter();
46:     //This time, we want to draw in the
47:     //window, so we let the this-pointer
48:     //define our drawing-area:
49:     paint->begin( this );
50:     //We use the drawPicture() function
51:     //to draw QPicture object:
52:     paint->drawPicture( pic );
53:     paint->end();
54: }
55:
56: void main( int argc, char **argv )
57: {
58:         QApplication a(argc, argv);
59:         MyMainWindow w;
60:         a.setMainWidget( &w );
61:         w.show();
62:         a.exec();
63: }
```

这里，在第 36 行，将一个简单的矩形存储到 file.pic 文件中。这样做，你在任何时候打开这个文件，并将其内容显示在屏幕上。第 42 到 53 行实现一个 paintEvent()函数，它用于在窗口上显示矩形。如果你跳过这一步，窗口将显示为空，只有 QPainter 的输出被保存到文件（如图 15-2 所示）。

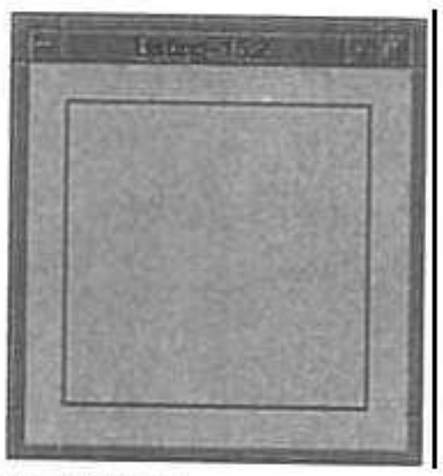


图 15-2　具有一个黑色矩形的窗口，这个矩形被保存到一个文件中

如前所述，你现在能够在任何时候装载文件 file.pic，并将它显示在屏幕上。为了这样做，需要使用 QPicture::load()函数：

```
QPicture pic;
pic.load( 'file.pic' );
QPainter paint;
Paint.begin( this );
Paint.drawPicture( pic );
Paint.end( );
```

这几行完成所有工作：装载文件，并将它显示在你定义的部件里。你可在程序的 paintEvent()函数中实现这几行代码。

15.2.2　所支持的图像格式

装载和显示所支持（也就是说，对这些格式你不需要使用其他额外库）的图像格式并

不困难。只需为图像创建一个 QPixmap 对象，并使用 QPainter::drawImage()函数绘制它即可。程序清单 15-3 给出一个例子：

程序清单 15-3　　打开和显示图像

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qpainter.h>
4: #include <qpixmap.h>
5:
6: class MyMainWindow : public QWidget
7: {
8: public:
9:     MyMainWindow();
10: private:
11:     void paintEvent( QPaintEvent* );
12: };
13:
14: MyMainWindow::MyMainWindow()
15: {
16:     resize( 150, 120 );
17: }
18:
19: void MyMainWindow::paintEvent( QPaintEvent* )
20: {
21:     //Create a QPixmap object for the
22:     //trolltech.bmp image:
23:     QPixmap image( "trolltech.bmp" );
24:
25:     QPainter paint;
26:     paint.begin( this );
27:     //Paint the image with its top-left
28:     //corner at horizontal position 11 and
29:     //vertical position 13:
30:     paint.drawPixmap( 11, 13, image );
31:     paint.end();
32: }
33:
34: void main( int argc, char **argv )
35: {
36:     QApplication a(argc, argv);
37:     MyMainWindow w;
38:     a.setMainWidget( &w );
39:     w.show();
40:     a.exec();
41: }
```

正如你在第 23 行所看到的，这里不需要指定所装载文件的图像格式，QPixmap 类能够自动进行判别，它首先在文件开头读取一小块，据此就能够判断它所使用的格式。因此，不必关心实际所使用的格式。而只要知道文件名即可（第 23 行）。

在这个例子中，使用一个叫做 trolltech.bmp 的 BMP 图像文件（第 23 行）。可以在 Qt 分发程序的 examples/widgets/目录中找到这个文件。该图像显示结果如图 15-3 所示。

如果程序具有编辑所装载图像功能，你很可能需要将编辑结果进行保存。所幸的是，这一工作也非常简单。只需调用 QPixmap::save()函数。这个函数有两个参数：第 1 个为字

符串，表示所存储的文件名称，第 2 个参数也是一个字符串，它表示存储文件所使用的图像格式。然而，最简单的办法是用 QPixmap::imageFormat()函数作为第 2 个参数。这个函数返回 QPixmap 对象所代表图像的图像格式。因此，如果需要将图像存储到/home/user/image.bmp 文件，并使用 QPixmap 对象 pixmap 所使用的图像文件格式，则应该按以下语法调用函数：

```
pixmap->save( "/home/user/image.bmp", pixmap->imageFormat( ) );
```



图 15-3　用 QPixmap 和 QPainter 类所显示的 BMP 图像

Qt 库不支持 JPG 图像格式。但是，可以用其他方式使用这种格式。为此，程序中需要包含 qimageio.h 头文件，并调用 qInitJpegIO()函数，之后将程序与 JPG 库一起链接。在 Linux/UNIX 系统下，通过向编译器添加-ljpeg 参数来链接 JPG 库（当然，必须在安装了 JPG 库后才能这样做）。更多信息请参看 Qt Reference Document。

15.3　QPainter 转换函数

QPainter 提供多个图像格式转换函数，这些函数的使用非常直观。因此，这一节不给出完整的例子，而只用几行代码加以说明。

15.3.1　图像缩放

QPainter::scale()函数能够缩放图像。它带两个 double 型参数。第 1 个参数表示水平方向缩放系数，第 2 个参数表示垂直方向缩放系数。图 15-4 显示出一个小矩形，图 15-5 为同一矩形在调用 scale(1.2, 1.2)函数（使矩形放大 0.2 倍）后的显示结果。

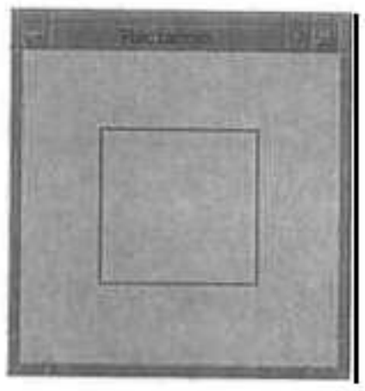


图 15-4　缩放前的矩形



图 15-5　缩放后的矩形

如你所看到的，它整个图像被放大，而不仅仅是其中矩形。因此，看起来好像是矩形被移向窗口的右下角。

15.3.2　图像剪切

使用 QPainter::shear()函数能够剪切图像。图 15-6 所显示出的图像为图 15-4 中的矩形在调用 shear(1.2, 1.2)函数后的显示结果。

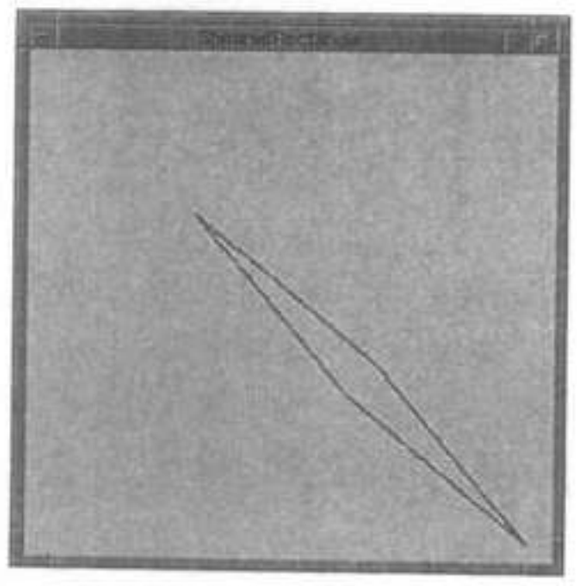


图 15-6　图 15-4 中的矩形在调用 shear(1.2, 1.2)后的显示结果

两个 double 参数表示你要剪切的坐标系统。

15.3.3　图像旋转

用 QPainter::rotate()函数能够旋转图像。只需用图像旋转度数作为还函数的参数即可进行旋转。图 15-7 显示出图 15-4 中的矩形在调用 rotate(30)函数后的显示结果。

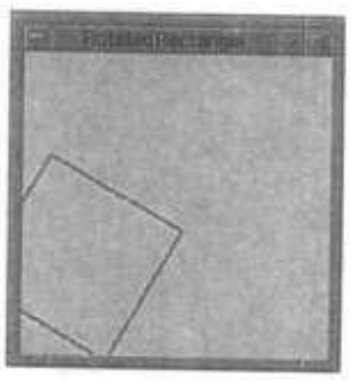


图 15-7　图 15-4 中的矩形在调用 rotate(30)后的显示结果

注意，整个图像被旋转，而不只是矩形。

15.3.4　图像平移

用 QPainter::translate()函数能够平移图像（移动图像的坐标）。它带两个参数：第一个参数表示水平方向平移的像素数，第二个参数表示垂直方向平移的像素数。在图 15-8 中，它调用 translate(-20, -30)对图 15-4 中的图像进行平移。

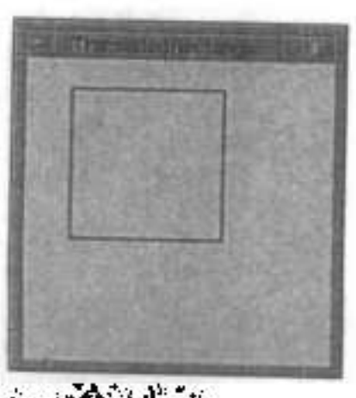


图 15-8　图 15-4 中的矩形在调用 translate(-20, -30)后的显示结果

如你所看到的，矩形被向左移动 20 个像素，向上移动 30 个像素。

15.3.5　改变视窗

用 QPainter::setViewport()函数能够设置矩形绘图区域的大小（以像素为单位）。这样做，可以限制用户只能在绘图设备的指定区域内绘图。缺省时，矩形大小设置为与绘图设备相同的尺寸。图 15-9 显示出图 15-4 中的图像在调用 setViewport (10, 10, 40, 40)函数后的显示结果。



图 15-9　图 15-4 中的矩形在调用 setViewport (10, 10, 40, 40)后的显示结果

当你在一个区域周围已经绘制边框，并想继续在这个区域内绘图而不触及边框时，这个函数就很有用。

15.3.6　设置窗口大小

使用 QPainter::setWindow()函数能够设置绘图设备尺寸。这个函数具有 4 个参数，它们定义一个代表绘图区域的矩形。前两个参数定义矩形的左上角，后两个参数定义矩形的右下角。图 15-10 中所显示的图像为对上面例子中的图像调用 setWindow(30, 30, 140, 140)函数后的显示结果。



图 15-10　这里将绘图设备缩小，因此矩形看起来比以前大

如你在图 15-10 中所看到的，缩小绘图设备的效果好像将该区域进行放大。相反，增大设备绘图区域，矩形将看起来比以前更小。图 15-11 为调用 setWindow(-20, -20, 140, 140)函数后的显示结果。

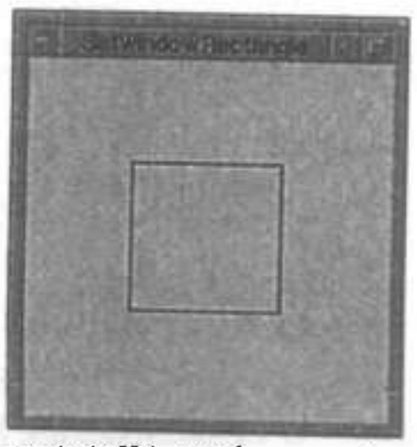


图 15-11　这里增大设备绘图区域，因此矩形看起来比以前更小

15.4　小　结

QPainter 的功能非常强大。但是，对于程序设计来讲，功能强大常常意味着复杂。QPainter 类也是这样。这个类很大，它具有很多成员，并且每个成员又具有很多选项。幸运的是，在这一学时和第 9 学时中所学习的内容已经完全可以满足大多需要。其他未学到的内容，可参看 Qt Reference Document。

为了在 Qt Reference Document 中查找到合适的功能，你至少应该对所要查找的操作有一点思路。如果你使用过其他图形程序，如 The Gimp 和 Photoshop，这就不会有任何问题。如果没有，这个任务变得有点困难，唯一的选择就是去测试各种不同的功能，看看它们的效果。记住，实践是最好的学习方法，并且测试也非常有趣！

15.5　问题与答案

问：重新编译 Qt 非常耗时！有没有办法能够加速这一过程？

答：不能加速实际编译。但是，如果你的存储空间足够，可以在首次编译 Qt 后将.o 文件保留在硬盘上。之后，如果需要重新编译 Qt 库（例如，增加对 GIF 的支持），这一过程将大大加快。当运行 make clean 命令时，.o 文件被删除。因此，如果你有足够的空间，可跳过这一步。

问：我试着在运行 GIF 动画期间改变其大小，怎样才能使 QLabel 对象尺寸随着动画大小一起改变？

答：这很简单。首先，创建一个改变标签大小的槽：

```
void MyMainWindow::resizeLabel( const QSize &size )
{
    label->resize( size );
}
```

之后，调用下面函数将该槽告诉动画：

```
movie->connectResize( this, SLOT( resizeLabel( const QSize& ) ) );
```

现在，标签大小就能够随着动画而改变。

问：用 QPainter 的转换函数对图像做几次转换后，我想撤消这些转换操作，恢复图像原来的样子。有什么简单的方法能够实现这一点吗？

答：是的，有。只需调用 QPainter::resetForm()函数，所有的转换操作都将被撤消。

问：我能够像打开其他 GIF 图像那样使用 QPixmap 对象打开 GIF 动画吗？

答：实际上可以这样做。但是你只能看到动画中的第一帧。因此，动画将不成为动画，而是一幅图像。

15.6　作　业

完成下面的问题和练习将有助于你牢记这一学时中所学到的图形知识。

15.6.1　测验

1. 在显示 GIF 动画之前你需要做的第一件事是什么？
2. 有什么方法能够加速动画吗？
3. 有什么方法能够读取动画的当前速度吗？
4. Qt 是否有它自己的图像格式？
5. 为什么需要使用 Qt 图像格式？
6. 在 Qt 中可以使用 JPG 图像吗？

15.6.2　练习

1. 修改程序清单 15-2 中的 paintEvent()函数，使它能够直接从磁盘读取文件，而不是使用 QPicture 对象。
2. 查找一个能够改变大小的 GIF 动画（如果你没有这类动画，可以访问 http://dir.yahoo.com/Arts/Visual_Arts/Animation/Computer_Animation/Animated_GIFs/ ）站点。在程序中使 QLabel 对象大小随着动画大小而改变（像在“问题与答案”中描述的那样）。
3. 编写一个圆（圆，而不是椭圆）绘图程序。使用 QPainter::rotate()函数旋转这个图像。旋转圆将看不出任何区别。如果有区别，说明为什么。

第 16 学时　程序间通信

实现一个程序内两个对象之间、或者是两个独立程序之间的通信功能将使程序变得更加容易使用。

剪贴板这--通信功能是利用一块内存区域临时存储程序中所用到的不同类型数据（通常为文本），它使一个程序内或不同程序之间的数据移动或拷贝变得非常简单。

拖放功能提供一种更加友好的数据移动或拷贝方法。你可以使用鼠标抓取一个对象，将它拖入另一个程序（或同一程序的另一部分），之后把它放在那里。

Qt 为提供一种非常简单的方法来实现剪贴板和拖放功能。这些功能在多数现代 GUI 应用程序中都能够找到，因此，你肯定也希望在应用程序中使用这些功能。这一课将教你怎样实现它们。

16.1　剪贴板

在 Qt 应用程序中能够很容易地实现剪贴板功能。实际上，一些 Qt 类带有可以使用的预定义剪贴板函数。

16.1.1　将剪贴板用于文本

多数情况下，剪贴板用于移动或拷贝文本。幸运的是，缺省时 QLineEdit 和 QMultiLineEdit 类均包含这一功能。当你创建这些类对象时，剪贴板功能被启动。你能够在 QLineEdit 和/或 QMultiLineEdit 对象之间拷贝或移动文本，组合键 Ctrl+C 用于拷贝，Ctrl+X 用于剪切或移动，Ctrl+V 用于粘贴。这些组合键在 UNIX 和 Windows 下的大多 GUI 应用程序中都是标准的，因此，你可能已经熟悉它们。

然而，你很可能想通过按钮或菜单来使用剪贴板。为此，需要将按钮或菜单连接到适当的槽。程序清单 16-1 是一个怎样实现这一功能的例子。

程序清单 16-1　　通过按钮使用剪贴板

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qmultilineedit.h>
4: #include <qpushbutton.h>
5:
6: class MyMainWindow : public QWidget
```

```
 6: class MyMainWindow : public QWidget
 7: {
 8: public:
 9:     MyMainWindow();
10: private:
11:     QMultiLineEdit *medit1;
12:     QMultiLineEdit *medit2;
13:     QPushButton *copy1, *cut1, *paste1;
14:     QPushButton *copy2, *cut2, *paste2;
15: };
16:
17: MyMainWindow::MyMainWindow()
18: {
19:     resize( 300, 300 );
20:
21:     //Create the Copy, Cut, and Paste buttons
22:     //for the left QMultiLineEdit object:
23:     copy1 = new QPushButton( "Copy", this );
24:     copy1->setGeometry( 10, 5, 40, 40 );
25:     cut1 = new QPushButton( "Cut", this );
26:     cut1->setGeometry( 55, 5, 40, 40 );
27:     paste1 = new QPushButton( "Paste", this );
28:     paste1->setGeometry( 100, 5, 40, 40 );
29:
30:     //Create the Copy, Cut, and Paste buttons
31:     //for the right QMultiLineEdit object:
32:     copy2 = new QPushButton( "Copy", this );
33:     copy2->setGeometry( 160, 5, 40, 40 );
34:     cut2 = new QPushButton( "Cut", this );
35:     cut2->setGeometry( 205, 5, 40, 40 );
36:     paste2 = new QPushButton( "Paste", this );
37:     paste2->setGeometry( 250, 5, 40, 40 );
38:
39:     //Create the left QMultiLineEdit object:
40:     medit1 = new QMultiLineEdit( this );
41:     medit1->setGeometry( 5, 50, 140, 245 );
42:
43:     //Create the right QMultiLineEdit object:
44:     medit2 = new QMultiLineEdit( this );
45:     medit2->setGeometry( 155, 50, 140, 245 );
46:
47:     //Connect the left buttons to the appropriate slots:
48:     connect( copy1, SIGNAL( clicked() ), medit1, SLOT( copy() ) );
49:     connect( cut1, SIGNAL( clicked() ), medit1, SLOT( cut() ) );
50:     connect( paste1, SIGNAL( clicked() ), medit1, SLOT( paste() ) );
51:
52:     //Connect the right buttons to the appropriate slots:
53:     connect( copy2, SIGNAL( clicked() ), medit2, SLOT( copy() ) );
54:     connect( cut2, SIGNAL( clicked() ), medit2, SLOT( cut() ) );
55:     connect( paste2, SIGNAL( clicked() ), medit2, SLOT( paste() ) );
56: }
57:
58: void main( int argc, char **argv )
59: {
60:     QApplication a(argc, argv);
61:     MyMainWindow w;
62:     a.setMainWidget( &w );
```

```
63:     w.show();
64:     a.exec();
65: }
```

这里，创建两个 QMultiLineEdit 对象（第 40、41、44、45 行），并为每个 QMultiLineEdit 对象创建 3 个按钮，分别用于控制剪切、拷贝和粘贴功能。左边 QMultiLineEdit 对象的按钮在第 23 到第 28 行中创建，右边 QMultiLineEdit 对象的按钮在第 32 到第 37 行中创建后。之后，在第 48、49、50 行和第 53、54、55 行将这些按钮的 clicked()信号分别连接到各自 QMultiLineEdit 对象的适当槽（copy()，cut()和 paste()）上。这样做的结果是可以通过这 6 个按钮在两个 QMultiLineEdit 对象之间拷贝、剪切和粘贴文本。该程序运行结果如图 16-1 所示。



图 16-1　在左边的 QMultiLineEdit 对象（medit1）中输入文本，之后将它拷贝到右边的 QMultiLineEdit 对象中（medit2）

现在，可以通过使用按钮或组合键访问剪贴板，在 medit1 和 medit2 之间实现文本拷贝和移动操作。

16.1.2　将剪贴板用于位图

QClipboard 类（它在 Qt 中处理剪贴板功能）也支持位图。但是，它没有定义位图拷贝、剪切和粘贴函数，因此，你必须自己定义。实际操作要比其听起来容易得多。程序清单 16-2 给出一个例子，说明怎样实现这一操作。

程序清单 16-2　　在剪贴板上使用位图

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qpixmap.h>
4: #include <qlistbox.h>
5: #include <qpushbutton.h>
6: #include <qclipboard.h>
7:
8: #include "clipboard.moc"
9:
```

```
10: //Since this class includes custom slots,
11: //remember to put the class definition in
12: //a file of its own and then use the MOC
13: //on it.
14: class MyMainWindow : public QWidget
15: {
16:     Q_OBJECT
17: public:
18:     MyMainWindow();
19: private:
20:     QListBox *left;
21:     QListBox *right;
22:     QPushButton *copy;
23:     QPushButton *cut;
24:     QPushButton *paste;
25: public slots:
26:     //We define one slot for each of the
27:     //clipboard functions:
28:     void copyXPM();
29:     void cutXPM();
30:     void pasteXPM();
31: };
32:
33: //The copyXPM() function will take the currently
34: //selected pixmap from the left QListBox object
35: //and copy it to the clipboard:
36: void MyMainWindow::copyXPM()
37: {
38:     const QPixmap temp( *(left->pixmap( left->currentItem())) );
39:     QApplication::clipboard()->setPixmap( temp );
40: }
41:
42: //The cutXPM() function works just like copyXPM().
43: //However, it also removed the item you selected
44: //to cut:
45: void MyMainWindow::cutXPM()
46: {
47:     const QPixmap temp( *(left->pixmap( left->currentItem())) );
48:     QApplication::clipboard()->setPixmap( temp);
49:     left->removeItem( left->currentItem() );
50: }
51:
52: //The pasteXPM() function inserts the pixmap
53: //currently in the clipboard into the right
54: //QListBox object:
55: void MyMainWindow::pasteXPM()
56: {
57:     right->insertItem( QApplication::clipboard()->pixmap() );
58: }
59:
60: MyMainWindow::MyMainWindow()
61: {
62:     resize( 200, 180 );
63:
64:     QPixmap pixmap( "home.xpm" );
65:
66:     left = new QListBox( this );
```

```
67:     left->setGeometry( 10, 40, 85, 130 );
68:     left->insertItem( pixmap );
69:
70:     right = new QListBox( this );
71:     right->setGeometry( 105, 40, 85, 130 );
72:
73:     copy = new QPushButton( "Copy", this );
74:     copy->setGeometry( 10, 10, 35, 20 );
75:
76:     cut = new QPushButton( "Cut", this );
77:     cut->setGeometry( 55, 10, 35, 20 );
78:
79:     paste = new QPushButton( "Paste", this );
80:     paste->setGeometry( 115, 10, 75, 20 );
81:
82:     //Connect the buttons to the slots we just created:
83:     connect( copy, SIGNAL( clicked() ), this, SLOT( copyXPM() ) );
84:     connect( cut, SIGNAL( clicked() ), this, SLOT( cutXPM() ) );
85:     connect( paste, SIGNAL( clicked() ), this, SLOT( pasteXPM() ) );
86: }
87:
88: void main( int argc, char **argv )
89: {
90:     QApplication a(argc, argv);
91:     MyMainWindow w;
92:     a.setMainWidget( &w );
93:     w.show();
94:     a.exec();
95: }
```

在这个程序中，显式调用 QClipboard 成员函数在两个 QListBox 对象之间插入和读取位图。为此，创建 3 个槽：copyXPM()、cutXPM()和 pasteXPM()。copyXPM()槽，在第 26 行到第 40 行定义，从左边的 QListBox 对象中提取当前选中的元素，并使用 QClipboard::setPixmap()函数将它拷入剪贴板（第 38 和 39 行）。cutXPM()槽，在第 45 行到第 50 行定义，看起来与 copyXPM()非常相似，差别是它使用 QListBox::removeItem()函数（第 49 行）同时删除所选中的元素。这个程序清单中的最后一个用户槽为 pasteXPM()（第 55 到 58 行）。它只包含单行代码（第 49 行），该代码读取剪贴板中的当前位图（使用 QClipboard::pixmap()函数），并使用 QListBox::insertItem()函数将它插入到右边的 QListBox 对象中。之后，在 MyMainWindow 构造函数中创建 3 个按钮：copy、cut 和 paste（第 73 到 80 行）。在第 83、84 和 85 行，这些按钮的 clicked()信号被连接到适当的槽。其结果是你可以将左边 QListBox 对象中的位图（在第 64 和 68 行创建和插入）拷贝和剪切到右边的 QListBox 对象中。图 16-2、16-3 和 16-4 说明该程序怎样工作。



如果你想使用图像而不是位图，使用 QClipboard::setImage()和 QClipboard::image() 函数代替 QClipboard::setPixmap() 和 QClipboard::pixmap()。

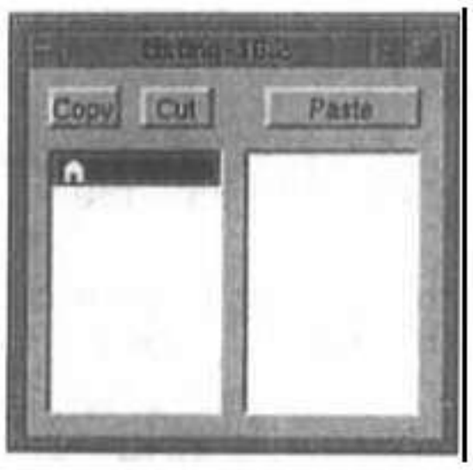


图 16-2　程序初始状态（当它第一次显示在屏幕上时）

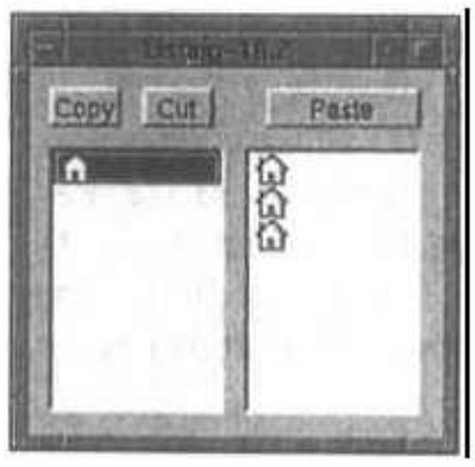


图 16-3　点击 1 次 Copy 按钮和 3 次 Paste 按钮

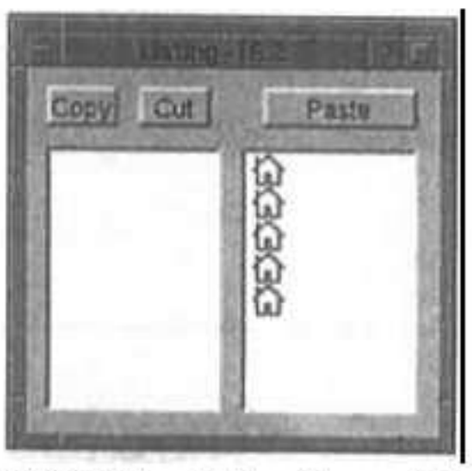


图 16-4　在程序初始状态下，点击 1 次 Cut 按钮和 5 次 Paste 按钮



注意，不能创建 QClipboard 对象。实际上，由于 QClipboard 的构造函数和析构函数是私有的，因此不能创建该类对象。QClipboard 对象被 QApplication 自动创建，程序可以通过 QApplication::clipboard()函数访问它们。

使用这个程序能够将任意数量的位图从左边拷贝到右边。你也可以使用剪切功能将左边的位图移到右边。但是，这个例子的功能非常弱，因为只能在一个方向（从左到右）上使用剪贴板。但实现另一个方向上的拷贝和移动操作一点也不困难。



实际上，有两个剪贴板函数使你能够将剪贴板用于其他类型的数据。它们是 QClipboard::setData()（将数据拷贝到剪贴板）和 QClipboard::Data()（从剪贴板提取数据）。这两个函数都只能处理 QMimeSource 类及其子类对象。因此，必须使使用剪贴板的类继承 QMimeSource 类。更多信息请参看 Qt Reference Document 中的“Drag and Drop”一部分。

16.2　实现拖放功能

如前所述，拖放功能对用户来说比剪贴板更加友好。但是，拖放功能也有缺点：实现起来比较复杂。幸运的是，在 Qt 中实现拖放功能并总不是那样复杂。

事实上，缺省时 QMultiLineEdit 类即具有拖放功能。你可以使用程序清单 16-1 来测试它。在左边区域中输入一些文本，用鼠标左键选中这些文本，然后点击左键并保持，将文本拖到右边区域。这时，只要放开鼠标左键，文本就将从左边区域拷贝到右边区域，它就是如此简单！

但是，如果你不满足于已经提供的拖放功能，就需要学习怎样实现用户拖放功能。因此，假设 QMultiLineEdit 类所提供的拖放功能不存在（像 Qt 1.4x 中那样），你需要自己来实现这些功能。 程序清单 16-3 向你介绍怎样实现这一功能。当你完全理解程序清单 16-3 后，再实现其他（更复杂的）拖放功能就毫无问题。

程序清单 16-3　　实现拖放功能

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qmultilineedit.h>
 4: #include <qdragobject.h>
 5:
 6: // ***** Start Definition Of Class MyDragSite *****
 7: //The following class is our drag site. We can drag
 8: //objects from this class to the drop site:
 9: class MyDragSite : public QMultiLineEdit
10: {
11: public:
12:         MyDragSite( QWidget *parent );
13: protected:
14:     //We need two functions for controlling
15:     //the drag:
16:         void mousePressEvent( QMouseEvent *event );
17:         void mouseMoveEvent( QMouseEvent *event );
18: };
19:
20: //This is a virtual function that is called when the
21: //mouse is pressed. You should add the code
22: //you want to be executed when the mouse is pressed
23: //into this function.
24: void MyDragSite::mousePressEvent( QMouseEvent *event )
25: {
26:         QMultiLineEdit::mousePressEvent( event );
27: }
28:
29: //This is a virtual function that is called when a
30: //drag is started.
31: void MyDragSite::mouseMoveEvent( QMouseEvent *event )
32: {
33:     //Here, a QTextDrag object os created. This is
34:     //our dragging object. We insert the text into
35:     //it (first argument) and set the dragging object
```

```
36:     //(the object we are dragging from) to this object.
37:     QTextDrag *drag = new QTextDrag( text(), this );
38:     //Start the dragging. By using dragCopy() instead
39:     //of dragMove(), the text will be copied, not moved.
40:     drag->dragCopy();
41:     QMultiLineEdit::mouseMoveEvent( event );
42: }
43:
44: MyDragSite::MyDragSite( QWidget *parent ) : QMultiLineEdit( parent )
45: {
46:     //We don't need anything in the constructor.
47: }
48: //***** End Definition Of Class MyDragSite
49:
50:
51: //***** Start Definition Of Class MyDropSite *****
52: //This is our drop class. We can drop things into
53: //objects of this class.
54: class MyDropSite : public QMultiLineEdit
55: {
56: public:
57:     MyDropSite( QWidget *parent );
58: protected:
59:     //Two functions are needed to control the drop:
60:     void dragEnterEvent( QDragEnterEvent *event );
61:     void dropEvent( QDropEvent *event );
62: };
63:
64: //dragEnterEvent should be implemented if the whole object
65: //shall accept drops. If just a part of the object shall
66: //accept drops, use the dragMoveEvent() instead. However,
67: //these functions are called when the object you are dragging
68: //is somewhere over this object. dragEnterEvent and
69: //dragMoveEvent() should be used to control if the dragging
70: //data can be received or not. In this case, we can only
71: //receive text.
72: void MyDropSite::dragEnterEvent( QDragEnterEvent *event )
73: {
74:     //The canDecode() function is used to check
75:     //whether the data can be decoded or not.
76:     //If the data is text, it can be decoded and
77:     //canDecode() returns TRUE.
78:     if( QTextDrag::canDecode( event ) )
79:     {
80:         if( QTextDrag::canDecode( event ) )
81:         {
82:             event->accept();
83:         }
84:     }
85: }
86:
87: //This virtual function is called when the data
88: //is accepted and it is dropped over the receiving
89: //object. In this case, we use the QTextDrag::decode()
90: //function to decode the QDropEvent object event into
91: //a QString object. This object is then inserted at the
92: //current position.
93: void MyDropSite::dropEvent( QDropEvent *event )
```

```
 94: {
 95:     QString text;
 96:
 97:     if( QTextDrag::decode( event, text ) )
 98:     {
 99:         int row, col;
100:         getCursorPosition( &row, &col );
101:         insertAt( text, row, col );
102:     }
103: }
104:
105: MyDropSite::MyDropSite( QWidget *parent ) : QMultiLineEdit( parent )
106: {
107:     //We don't need anything to be done in the constructor.
108: }
109: //***** End Definition Of Class MyDropSite *****
110:
111: void main( int argc, char **argv )
112: {
113:         QApplication a(argc, argv);
114:         QWidget w;
115:     w.resize( 200, 200 );
116:
117:     //Create an object of the MyDragSite class:
118:     MyDragSite dragsite( &w );
119:     dragsite.setGeometry( 10, 10, 85, 180 );
120:
121:     //Create an object of the MyDropSite class:
122:     MyDropSite dropsite( &w );
123:     dropsite.setGeometry( 105, 10, 85, 180 );
124:
125:         a.setMainWidget( &w );
126:         w.show();
127:         a.exec();
128: }
```

这个程序清单具有很好的注释。此外，这里再对其基本细节加以介绍：

● 当在一个对象上按下鼠标左键时，mousePressEvent()（第 24 到 27 行）函数被调用。如果你想在用户按下鼠标按钮时做其他事情，应该将代码添加到这个函数内；

● 当用户开始移动鼠标时（鼠标左键被一直按下），mouseMoveEvent()（第 31 到 42 行）函数被调用。如果想使其他事情在这时发生，应将适当的代码添加到这个函数；

● 当光标定位到接收对象上时，该对象的 dragEnterEvent()（第 72 到 85 行）函数被调用（当然，只有当这个对象具有该函数时）。应该向这个函数添加代码，决定是否解码（接收）数据。通常，鼠标指针指示出数据是否能够放下。注意，如果不想使整个部件能够接收数据，可以使用 dragMoveEvent()函数代替 dragEnterEvent()。关于怎样使用 dragMoveEvent()函数的更详细内容请参看 Qt Reference Document 中“QWidget”部分；

● 最后，如果数据能够解码和放置，dropEvent()函数被调用（第 93 到 103 行）。这个函数所包含的代码能够解码所拖对象，并将结果插入到它被放置的位置。

注意，Qt 确保上面所讨论的这些函数能够在正确的时刻被调用。这些你不用担心。

关于拖放功能的更多信息请参看 Qt Reference Document 中的“Drag and Drop”部分。关于鼠标事件的更多信息请参看 Reference Document 中的“QWidget”部分。

16.3　小　结

剪贴板和拖放功能在多数 GUI 应用程序中已经成为标准。因为它们使很多操作变得更加简单，你应该把二者都看作是应用程序的很好功能。作为一个开发者，应该努力创建尽可能容易使用的好应用程序，因此，在多数情况（尽管不是所有情况）下多花一些时间来实现这些功能是值得的。

很自然，剪贴板功能非常直观。如果你想为用户数据类型创建剪贴板功能，它要不了几行代码就可以实现。

相反，拖放功能绝不会被认为“直观”。但是，只要确实理解 QWidget 中（并被许多其他 Qt 所继承）所定义的鼠标事件函数，你就能够很好地实现它。查找应该调用哪个函数的好方法就是使用 qDebug()函数（将在第 23 学时“调试技术”中介绍）。通过在一个你不确定的函数中使用适当的字符串做参数来调用 qDebug()，当函数被调用时字符串将被输出到 stdout（或者标准输出——你的监视器），就能够很容易地看到所发生的事情。你应该阅读 Reference Document 中的“Drag and Drop”部分，它包含一些有趣的内容。

16.4　问题与答案

问：是否有办法为剪贴板定义我自己的组合键？

答：是的，QWidget::keyPressEvent()函数处理按键事件。在那里，你可以定义当某个键按下时所发生的事件。更多信息请参看 Reference Document。

问：有办法清除剪贴板中的数据吗？

答：是的，只需调用 QClipboard::clear()函数（qApp->clipboard->clear()）。

问：我对拖放功能有点困惑。我想将一个 Qt 应用程序文本区域内的文本拖到另一个应用程序的一个文本区域。需要做什么事情吗？

答：没有，绝对不需要你做其他额外的事情。

16.5　作　业

完成下面的问题和练习将有助于你牢记这一学时中所学到的知识。通过完成这些练习，你能够确实理解怎样用 Qt 实现程序的通信功能。

16.5.1　测验

1. 怎样做才能使用标准组合键实现文本的拷贝和移动操作？
2. 用标准 Qt 剪贴板函数能够实现图像的剪切和粘贴吗？
3. 如果想使用剪贴板，应该基于哪个类？
4. 只让一个部件的某个区域能够接收拖放对象，这可能吗？

5. 当将对象放置在一个能够接收对象的区域时，应该调用哪个函数？
6. 使一些部件只能从中拖动对象而不能在其上放置对象，或者使另一些部件只能放置对象而不能从中拖动对象。这可能吗？

16.5.2　练习

1. 进一步开发程序清单 16-2，使它能够在另一个方向上传递位图。
2. 为程序清单 16-3 添加一个功能，使对象在移动 3 个像素之后才开始拖动（提示：使用 QmouseEvent::pos()函数）。
3. 试着编写一个能够接收拖放对象的部件，但只在指定的区域内（例如，部件左上角一个 100×100 像素的正方形）。

第四部分

Qt 编程技巧

第 17 学时　编写 KDE 应用程序的第 1 课

第 18 学时　编写 KDE 应用程序的第 2 课

第 19 学时　使用 Qt 的 OpenGL 类

第 20 学时　创建 Netscape 插件

第 17 学时　编写 KDE 应用程序的第 1 课

如果你是一个 UNIX 用户（或所有其他变体版本），很可能听说过 KDE 桌面。这是一套程序和库，它给 UNIX 用户提供一个更加友好的桌面，与 Microsoft Windows 98 所提供的桌面非常类似。KDE 是一个全功能的图形工作环境，具有你所需要的所有部件。与 Windows 一样，KDE 有它自己的文件管理器，并可用做 Web 浏览器。它包含文本编辑器、图形软件、各种网络实用程序（用于 Internet 和局域网）、游戏和其他很多有用项目。一个特殊的办公类程序包也正在开发之中。

你可能了解到 KDE 是使用 Qt 开发的。但是，KDE 有它自己的 API，它是由基于 Qt 类所构造的类组成。使用这些 KDE 类，要遵守各种 KDE 标准，使你的应用程序能够适合 KDE 系列软件。为了更好地了解 KDE 和 KDE 标准，请参看图 17-1，它显示一个基本的 KDE 桌面。

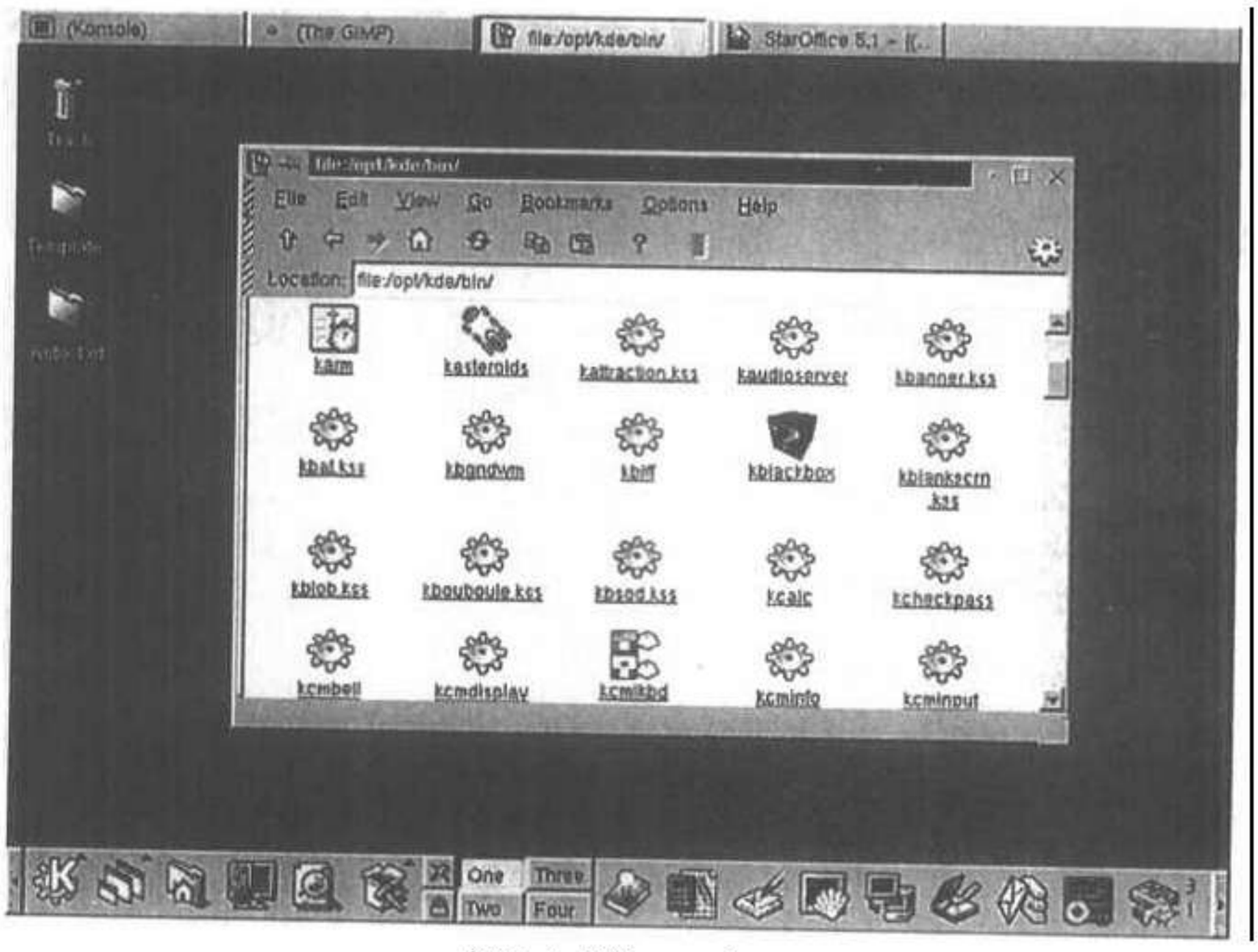


图 17-1　标准 KDE 桌面

在图 17-1 中，屏幕下端为 KDE 面板，从这里能够启动所有 KDE 程序和改变所使用的虚拟桌面。屏幕上端为任务栏，与 Windows 95/98 中类似。在屏幕的中间，正运行着 KDE

文件管理器。

在这一学时和下一学时里，将介绍 KDE 程序设计。这两学时将讨论纯 Qt 程序设计和 KDE 程序设计之间的差别，并给出几个简单的 KDE 程序设计例子。

17.1 KDE 程序设计基础

你需要做的第一件事是学习 KDE 程序设计基本知识。在这一节，将创建一个非常简单的 KDE 程序，讨论构造这个程序所基于的 KDE 类，并将这个程序与常规 Qt 应用程序进行比较。



这一学时和下一学时中屏幕拷贝图像为 KDE 窗口管理器，而不是其他学时中所使用的 Motif 窗口管理器。建议你在 KDE 环境下开发，以访问 KDE 库所提供的所有功能。

17.1.1 安装 KDE

当然，在开始开发 KDE 应用程序之前需要做的第一件事是安装 KDE 及其开发库。

如果你正在使用 Linux/UNIX 分发程序，KDE 很可能就包含在分发程序 CD 中，你可以像安装分发程序中的其他程序一样来安装 KDE 包。但是，要确保安装开发库。这些包在它们的文件命中通常有 devel 字。

如果 KDE 未包含在 Linux 分发程序中，或者是在使用不带 KDE 的变种 UNIX 时，你可以很容易地从www.kde.org下载所需文件，在这个站点上，你还能够查找到怎样在系统上安装（和编译，如果需要的话）包方面的信息。

17.1.2 编写第一个 KDE 程序

我们从一个非常简单的 KDE 程序开始，它类似于第 1 学时“Qt 简介”中所讨论的第一个 Qt 程序。阅读程序清单 17-1，看你能否找出它们之间的差别。

程序清单 17-1 一个简单的 KDE 程序

```
1: #include <kapp.h>
2: #include <ktmainwindow.h>
3:
4: int main( int argc, char **argv )
5: {
6:     KApplication a( argc, argv );
7:
8:     KTMainWindow *w = new KTMainWindow();
9:     w->resize( 200, 100 );
10:    a.setMainWidget( w );
11:    w->show();
12:
13:    return a.exec();
14: }
```

如你所看到的，这个程序很像 Qt 程序，尽管它有一些重大的差别。首先，它不包含任何 Qt 头文件。这是因为你不直接使用所有的 Qt 类（KDE 将处理这些）。相反，它包含两个 KDE 头文件，kapp.h（第 1 行）和 ktmainwindow.h（第 2 行），kapp.h 定义 KApplication 类（第 6 行创建一个 KApplication 对象），ktmainwindow.h 定义 KTMainWindow 类（第 8 行创建一个 KTMainWindow 对象）。

KApplication 对象在 KDE 程序中的功能与 QApplication 对象在纯 Qt 程序中的功能相同。但是，KApplication 对象具有其他功能，它使程序遵守 KDE 标准。Kapplication 首先创建一个 KConfig 对象，它用于访问 KDE 配置项。KApplication 也提供各种 KDE 资源，如菜单项、加速键、帮助请求和会话管理。

KTMainWindow 类创建标准的 KDE 主窗口。它提供工具栏和状态栏。KTMainWindow 也自动设置图标、最小图标和应用程序标题（从 QApplication 所接收的信息）。

之后在 main()函数中创建这些类对象。如你所看到的，可以像 Qt 类一样使用它们。现在，准备编译这个程序。怎样编译依赖于环境变量设置。这里是一种安全可靠的方法：

```
$ g++ -I$KDEDIR/include -L$KDEDIR/lib -lqt -lkdecore -lkdeui 17lst01.cpp
        -o listing-17.1
```

这能够确保编译器在适当的目录中查找到头文件和库。注意，要同时链接 kdecore 和 kdeui 库。访问 KApplication 需要链接 kdecore，访问 KTMainWindow 需要链接 kdeui。该程序显示如图 17-2 所示。



图 17-2　一个非常简单的 KDE 程序

如你所看到的，这可能是你能够创建的最简单的 KDE 程序。它没有任何按钮、菜单、状态栏。实际上，它也没有任何实际用途。

17.1.3　添加按钮、菜单、工具栏和状态栏

如前所述，KTMainWindow 提供添加按钮、菜单、工具栏和状态栏等功能，这些项目符合 KDE 标准。程序清单 17-2 给出一个例子：

程序清单 17-2　　具有菜单、工具栏、状态栏和按钮的 KDE 程序

```
 1: #include <kapp.h>
 2: #include <ktmainwindow.h>
 3: #include <kkeydialog.h>
 4: #include <kstatusbar.h>
 5: #include <kmenubar.h>
 6: #include <qpopupmenu.h>
 7: #include <ktoolbar.h>
 8:
 9: class MyKDEProgram : public KTMainWindow
10: {
11: public:
```

```
12:     MyKDEProgram();
13: private:
14:     KKeyButton *exit;
15:     KStatusBar *sbar;
16:     KStatusBarLabel *slabel;
17:     KMenuBar *menu;
18:     QPopupMenu *file;
19:     QPopupMenu *help;
20:     KToolBar *tbar;
21: };
22:
23: MyKDEProgram::MyKDEProgram() : KTMainWindow()
24: {
25:     resize( 250, 150 );
26:
27:     //Create a keyboard-like button:
28:     exit = new KKeyButton( "exit", this );
29:     exit->setGeometry( 95, 70, 80, 40 );
30:     exit->setText( "Exit" );
31:
32:     //Create a status bar:
33:     sbar = new KStatusBar( this );
34:     //Add a label to the status bar:
35:     slabel = new KStatusBarLabel( "This is a status bar", 1, sbar );
36:     setStatusBar( sbar );
37:
38:     //Create the file-menu:
39:     file = new QPopupMenu();
40:     file->insertItem( "&Exit", kapp, SLOT( quit() ) );
41:     //Create the help-menu:
42:     help = kapp->getHelpMenu( TRUE, "Simple KDE Program" );
43:     //Create the menu bar and add the two menus to it:
44:     menu = new KMenuBar( this );
45:     menu->insertItem( "&File", file );
46:     menu->insertItem( "&Help", help );
47:     //Set menu as the menubar of this KTMainWindow object:
48:     setMenu( menu );
49:
50:     //Create a toolbar:
51:     tbar = new KToolBar( this );
52:     //Insert three icons to the tool bar:
53:     tbar->insertButton( QPixmap( "fileopen.xpm" ), 0 );
54:     tbar->insertButton( QPixmap( "filesave.xpm" ), 1 );
55:     tbar->insertButton( QPixmap( "fileprint.xpm" ), 2 );
56:     //Set tbar as the status bar of this KTMainWindow object:
57:     addToolBar( tbar );
58:
59:     //Connect the KKeyButton to the quit() slot of kapp:
60:     connect( exit, SIGNAL( clicked() ), kapp, SLOT( quit() ) );
61: }
62:
63: int main( int argc, char **argv )
64: {
65:     KApplication a( argc, argv );
66:     MyKDEProgram w;
67:     a.setMainWidget( &w );
68:     w.show();
```

```
69:     return a.exec();
70: }
```

KDE 类的使用与 Qt 类的使用非常类似。二者主要差别是 KDE 类遵守 KDE 标准，这是合乎需要的。这里，用户类基于 KTMainWindow 类（第 9 行），对于所有 KDE 项目，推荐都采用这种方法。

在 MyKDEProgram 类中，创建几个普通的 KDE 类对象。在构造函数中（第 23 行到 61 行），首先设置窗口大小（第 25 行）。这与 Qt 程序的设置方法相同——使用 resize()函数。之后，用 KKeyButton 类创建键盘状按钮（第 28、29、30 行）。如果这里使用 QPushButton 按钮也能很好工作，但因为这一学时是讲解 KDE，所以要尽可能多地坚持使用 KDE 类。注意，需要调用 KKeyButton::setText()函数来设置按钮标签（第 30 行）。

在此之后，创建一个 KStatusBar 类对象（第 33 行），它在窗口的下端创建一个状态栏。之后，设置状态栏标签（第 35 行），并使用 setStatusBar()函数将状态栏告诉 KTMainWindow（第 36 行）。注意，这个状态栏一直显示相同的文本。但由于状态栏的功能是将用户当前所指项目通知用户，所以，实现这个状态栏的用处不大。其代码应当在适当的鼠标事件函数中实现，以使状态栏中的文本能够随着鼠标当前所指内容而改变。

之后，使用 Qt 类 QPopupMenu 创建两个菜单：File 和 Help（第 39、40 和 42 行）。定义 Help 菜单行非常有趣：

```
help = kapp->getHelpMenu( TRUE, "Simple KDE Program" );
```

这一行使 KApplication 类处理菜单定义。TRUE 参数说明希望在 Help 菜单中包含一个 About QT 项。其后字符串为选中 About <程序名>菜单项后所显示的文本信息。尽管可以使用 QMenuBar 对象插入这些菜单，但是，这里使用 KMenuBar 对象（第 44、45、46 行），使这个程序尽可能与 KDE 兼容。最后，调用 KTMainWindow::setMenu()函数（第 48 行）将菜单栏告诉 KTMainWindow。

使用 KToolBar 类创建工具栏（第 51 行），之后用 KMenuBar::insertItem()函数向工具栏中插入 3 个图标（第 53、54 和 55 行），最后，通过调用 KTMainWindow::addToolBar()函数将工具栏告诉 KTMainWindow（第 57 行）。

MyKDEProgram 构造函数的最后一个任务是将 KKeyButton 连接到 KApplication 对象的 quit()槽。注意，使用指针 kapp 访问还未创建的 KApplication 对象，kapp 的工作方式与 qApp 完全相同。

现在，按照前面介绍的方法编译该程序。图 17-3 为该程序的运行结果。



图 17-3　一个相当复杂的 KDE 程序，与其他标准 KDE 程序一样，它具有菜单、工具栏、按钮和状态栏

如你所看到的，按钮看起来有一点不同，并且菜单栏和工具栏的左边有一些条纹区域。这些区域提供相当有用的功能，如果你用鼠标左键抓取任一个区域，能够将该栏拖放到窗口

上的其他位置，或者甚至将它放置在窗口外部。图 17-4 即是一个例子。

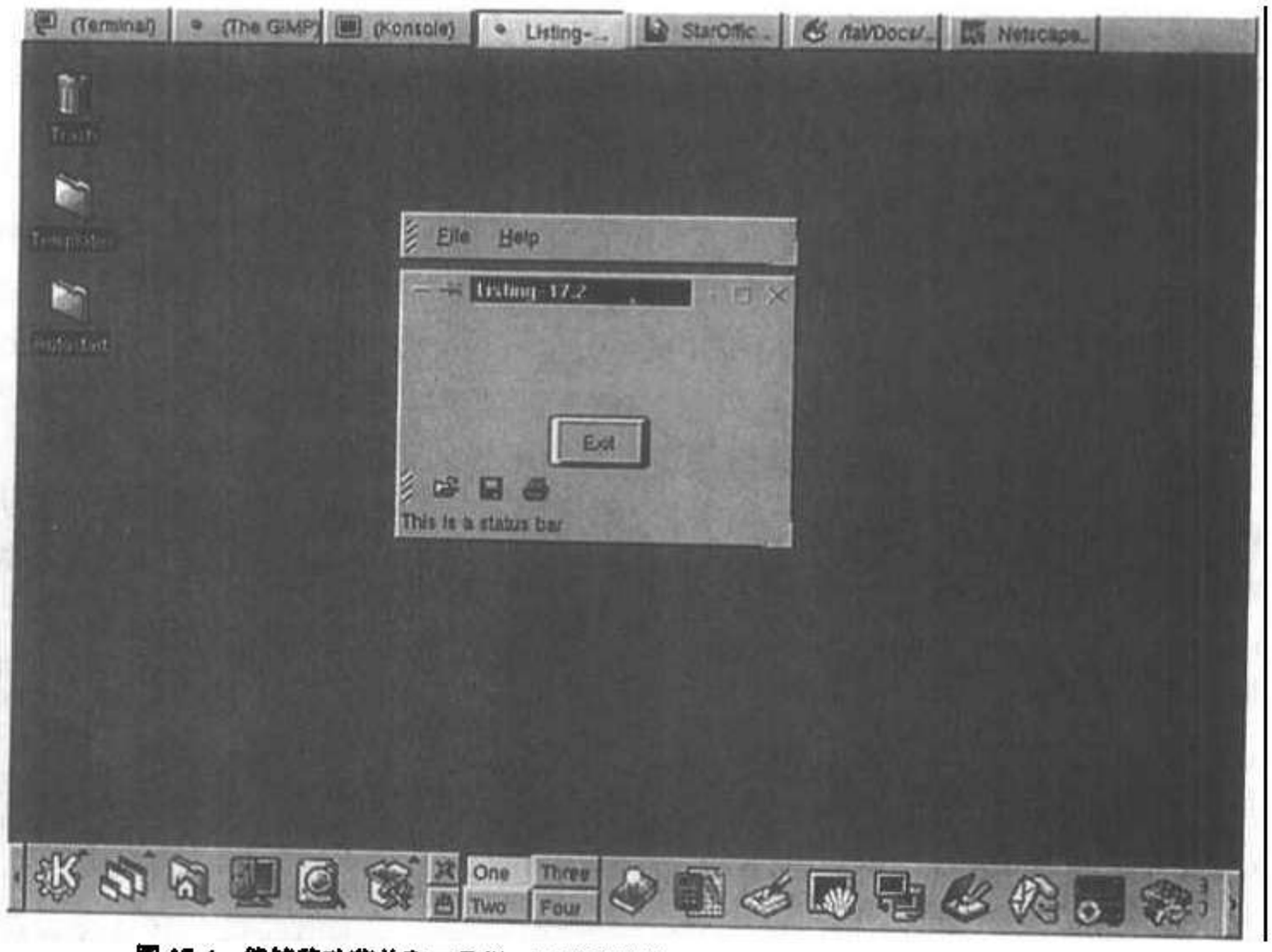


图 17-4 能够移动菜单和工具栏。工具栏被移到窗口的底部，菜单栏被移到窗口外部

现在来看 Help 菜单，它由 KApplication::getHelpMenu()创建。这个菜单如图 17-5 所示。

如你所看到的，这个菜单具有 3 个菜单项：Contents，About Listing-17.2…和 About KDE…。这 3 个菜单项由 KApplication::getHelpMenu()创建。如果程序文档存在，并被放置在适当的位置，第 1 个菜单项，Contents，将打开这些文档。如果点击该项，将在 KDE 程序文档的标准位置搜索程序文档。为此，需要将文档放置在那里。现在，我们来点击该项，看看会发生什么事情。图 17-6 为所显示出的窗口。

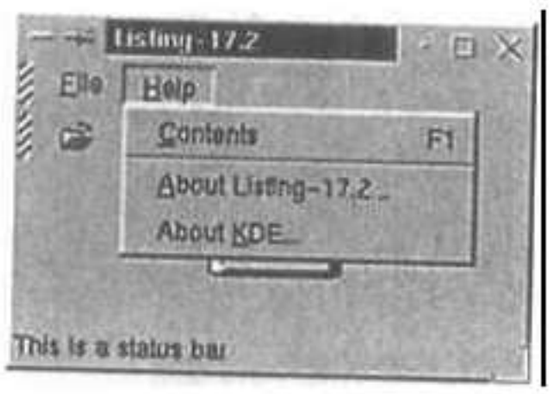


图 17-5 KApplication::getHelpMenu()创建的 Help 菜单



图 17-6 当点击 Contents 菜单项而没有查找到文档时所显示的消息窗口

如你在窗口文本中所看到的，程序试图在/opt/kde/share/doc/HTML/default/listing-17.2 目录中查找 index.html 文件。你可能想知道为什么是这个目录呢？这是因为：在这里 KDE 被

安装在/opt/kde（$KDEDIR 被设置为/opt/kde）目录下，并且程序名称为 listing-17.2。缺省时，在$KDEDIR/share/doc/HTML/default/<程序名>目录下搜索 KDE 程序文档。如果将 KDE 安装到/usr/local/kde 下，并且程序文件被命名为 kdeprogram，那么，index.html 的搜索目录将变为/usr/local/kde/share/doc/HTML/default/kdeprogram。



也可以通过 KApplication 构造函数的第 3 个参数来设置程序名称。因此，如果想调用应用程序 MyProgram，你应该做以下调用：

```
KApplication a(argc, argv, "MyProgram");
```

如果点击第 2 个菜单项，About Listing-17.2…，所显示的窗口如图 17-7 所示。



图 17-7　当点击 About Listing-17.2…菜单项时所显示的消息窗口

显示在这个窗口中的文本为传递给 QApplication::getHelpMenu()函数的第 2 个参数。通常这个窗口用于显示关于程序的简短信息。其中常包含作者名称和联系信息等。

第 3 项，About Listing-17.2…，为选项。但是，因为传递给 QApplication::getHelpMenu()函数的第 1 个参数为 TRUE，所以这里包含它。如果你点击该项，将显示 KDE 信息窗口（如图 17-8 所示）。

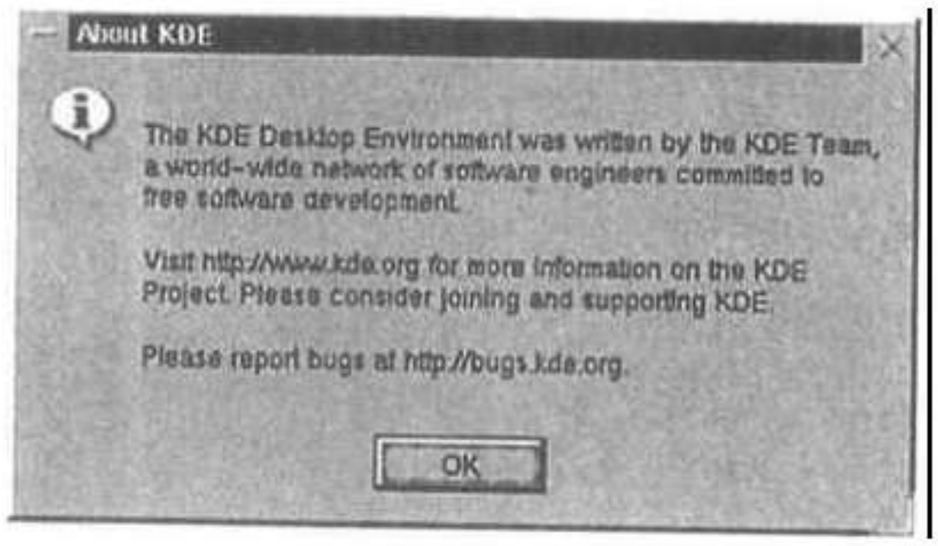


图 17-8　标准 KDE 信息窗口

程序员几乎都在他们的程序中包含 About KDE…选项。这是一个很好的习惯，希望你在程序中也这样做。

17.2　使用 KDE 的 HTML 功能特点

由于 Internet 的快速发展，很自然地，用于创建 Internet 最流行的服务——World Wide Web 的语言，也变得更加流行。近来，HTML 不仅用于创建 Web 站点，也用在很多其他方面，如文档（KDE 中也是这样）。

KDE 具有一个完整库，libkhtmlw，它可用在程序中，libkhtmlw 包含许多类，但是，唯

一你关心的类是 KHTMLWidget 类，其他类在 KDE 内部使用。为了在程序中显示 Web 页面，你需要创建 KHTMLWidget 对象，启动 HTML 解析，并用 HTML 代码填充 KHTMLWidget 对象。程序清单 17-3 给出一个例子：

程序清单 17-3　　简单的 HTML 浏览器

```
1: #include <kapp.h>
2: #include <ktmainwindow.h>
3: #include <html.h>
4: #include <qstring.h>
5:
6: class MyKDEProgram : public KTMainWindow
7: {
8: public:
9:     MyKDEProgram();
10: private:
11:     KHTMLWidget *mybrowser;
12: };
13:
14: MyKDEProgram::MyKDEProgram() : KTMainWindow()
15: {
16:     resize( 200, 200 );
17:
18:     QString htmlcode = "<html><head><title>My Web Browser</title></head>
19:                 <body bgcolor=\"#ffffff\"><center><h1>This is my
20:                 own WWW browser!</h1></body></html>";
21:
22:     mybrowser = new KHTMLWidget( this );
23:     mybrowser->resize( 200, 200 );
24:     mybrowser->begin();
25:     mybrowser->parse();
26:     mybrowser->write( htmlcode );
27:     mybrowser->end();
28: }
29:
30: int main( int argc, char **argv )
31: {
32:     KApplication a( argc, argv );
33:     MyKDEProgram w;
34:     a.setMainWidget( &w );
35:     w.show();
36:     return a.exec();
37: }
```

这个概念非常简单。首先，创建 KHTMLWidget 对象，并设置其大小（第 22 和 23 行）。之后，调用 KHTMLWidget::begin()函数启动 KHTMLWidget 对象（第 24 行）。如果需要，你可以输入一个 URL 地址作为该函数的参数，该页将被显示出来。在此之后，调用 KHTMLWidget::parse()函数启动 HTML 解析（第 25 行）。接着调用 KHTMLWidget::write()函数（第 26 行）。这个函数为 KHTMLWidget 对象提供一些 HTML 代码（在第 18、19、20 行 MyKDEProgram 构造函数的开始部分定义）。最后，调用 KHTMLWidget::end()函数通知 KHTMLWidget 对象 HTML 代码插入结束（第 27 行）。

当编译这个程序时，根据系统不同，所需要添加的链接参数也不相同。但是，下面是

一种相当安全的方法：

```
g++ -I$KDEDIR/include -L$KDEDIR/lib -lqt -lkdecore -lkdeui -lkhtmlw -lkimgio
-ljpeg -lpng -lgif -ltiff -ljscript 17lst01.cpp -o listing-17.1
```

如果在编译时出现未定义符号错误，则应查找哪个库中包含这些符号定义，并与它们进行链接。通常符号名称中已经指出库名。例如，如果一个未定义符号名中包含 java 或 JS 字符串，你可以猜想它是 Java 库中的一部分，或许是 libjs 或 libjscript。相反，如果链接器报告未找到它所链接的库，只要从列表中将该库删除，然后再重新链接一次。

在查找符号时你应该了解的一个有用工具是 nm 程序。它能够列出一个程序或库中的符号。运行 man nm 能够查看更详细内容。该程序界面如图 17-9 所示。



图 17-9　一个简单的 HTML 程序，它用 KHTMLWidget 类显示 HTML 页面

当然，KHTMLWidget 类还包含其他一些有用函数，这些函数如表 17-1 所示。

表 17-1　KHTMLWidget 成员函数

函　数	描　述
KHTMLWidget::print()	调用这个函数打印 HTML 页面
KHTMLWidget::getSelectedText()	这个函数返回用户所选择的文本（为一个 QString 对象）
KHTMLWidget::isTextSelected ()	如果文本被选中则返回 TRUE，否则，返回 FALSE
KHTMLWidget::getURL()	通过向这个函数传递一个指向 QPoint 对象的指针，来确定 QPoint 对象所代表点是否有一个 URL，如果有，它返回一个字符向量
KHTMLWidget::docWidth()	返回被解析的 HTML 代码的像素宽度
KHTMLWidget::docHeight()	返回被解析的 HTML 代码的像素高度
KHTMLWidget::getDocumeatURL()	返回表示 HTML 页面 URL 地址的 KURL 对象
KHTMLWidget::getSelectedFram()	返回指向一个 KHTMLView 对象的指针，该对象描述当前所选择帧（如果使用帧集合方式显示的话）
KHTMLWidget::isFrame()	如果当前所选择文档为帧则返回 TRUE，否则返回 FALSE
KHTMLWidget::setDefaultFontBase()	使用这个函数设置缺省字体大小，传递给它的参数应为 2 到 5 之间的整数
KHTMLWidget::setFixedFont()	与前一个函数的用法相同，但它设置的是固定字体，而不是缺省字体
KHTMLWidget::setDefaultFont()	调用这个函数改变缺省字体，输入你所想用的字体作为字符串参数
KHTMLWidget::setDefaultBGColor()	使用这个函数设置缺省背景颜色。用 QColor 对象做参数描述所选择使用的颜色
KHTMLWidget::setDefaultTextColors()	它带两个 QColor 对象做参数，第 1 个表示缺省文本颜色，第 2 个表示缺省链接颜色

实际上，还有很多函数，但这里所列出的是一些最常用的函数。$KDEDIR/include/html.h（它在 KDE 2.0 中很可能被修改为 khtml.h）中包含所有函数信息。但你可能发现有些函数并不令人感兴趣！

事实上，KHTMLWidget 是一个功能很强的工具。使用它在程序中能够很容易地显示具有 CSS（级联样式表）和 Java 的 HTML 页面。



在写这本书时，KDE 2.0 还没有发布。但是，在 KDE 2.0 中 KHTMLWidget 类将有很大改进。如果你使用的是这个版本，将发现 KHTMLWidget 更像一个全功能的 Web 浏览器，而不只是一个 HTML 文档显示部件。

17.3 小 结

现在，你对 KDE 项目有了一个基本的理解，并知道怎样进行 KDE 开发。在这一学时中你可能已经注意到，使用 KDE 类与使用 Qt 类非常相似。这可能使你产生怀疑是否真正需要再学习 KDE 库。当然，如果不打算使应用程序与 KDE 兼容，这是不需要的，但这是一件很好的事情。

KDE 使 UNIX/Linux 初学者能够更容易地学习使用系统。如果显现给初学者的是一个具有 KDE 这样的 GUI 界面，那么他们适应系统的机会就比提供给他们一个 UNIX/Linux 字符界面要大得多。如果你的应用程序遵守 KDE 标准，那些已经熟悉 KDE 的人对你的程序也会感到熟悉，他们能够很快学会使用它。如果有很多用户需要使用你开发的应用程序，它就会被包含到标准 KDE 分发程序中，并将因此使更多的人对 KDE 感兴趣。一般来说，对 KDE 项目所做出的贡献，也是对 Linux/UNIX 做贡献。因为，帮助使 KDE 桌面环境变得更好，就能够激起更多的人对这些平台感兴趣。

记住，当按照 KDE 进行开发时，不必对所有东西都使用 KDE 类（尽管这是可能的）。在多数情况下，只需使用 KApplication 和 KTMainWindow 类。对于其他内容，用标准 Qt 类就能够做得很好。

17.4 问题与答案

问：即使是严格按照前面的指导，我的编译器也不能找到 KDE 库或头文件，这是为什么？

答：首先，你需要确保 KDE 已经安装。之后查看 KDEDIR 外壳变量是否被正确设置。执行下面命令进行检查：

```
$ echo $KDEDIR
```

如果该命令的输出为你系统中 KDE 的安装目录，说明它是正确的。例如，如果 KDE 被安装在/opt/kde 下，KDEDIR 也应该被设置为/opt/kde。如果 KDEDIR 没有设置，你可以使用下面命令进行设置：

```
$ export KDEDIR=/opt/kde
```

但是，你很可能需要将它添加到启动文件中，例如，/etc/profile。

问：当使用 KHTMLWidget 类时，不能显示 HTML 文档，这是为什么？

答：要确保在适当的时候调用正确的函数，KHTMLWidget 对此非常敏感。

问：当按照这一学时中的指导编译一个使用 KHTMLWidget 的程序时，我的编译器报告它未找到图像库（libgif、libpng 或 libjpeg）之一。到哪里能够找到这些库？

答：这些库和很多其他 Linux 软件都可以在 www.freshmeat.net 中找到。此外，ftp://sunsite.unc.edu也是一个不错的站点。

问：当我调用 KHTMLWidget::print()函数时，我的打印机能够打印，但是它是些无法阅读的符号，这是为什么？

答：这个问题很可能不是你的程序引起的。在多数情况下，这是错误的打印机驱动程序或根本就没有打印机驱动程序所造成的。

17.5 作　业

完成下面的问题和练习将有助于你牢记这一学时中所学到的 KDE 程序设计基本知识。

17.5.1 测验

1. 什么是 KDE?
2. 当编写 KDE 程序时，应该使用哪个类代替 QApplication?
3. 什么是 kapp?
4. 在 KDE 程序中应该使用哪个类创建主窗口？
5. KApplication::getHelpMenu()有什么用途？
6. 在 KDE 程序中是否有显示 HTML 文档的好方法？

17.5.2 练习

1. 编写一个程序，创建一个 KHTMLWidget 对象，之后在其中显示 Qt Reference Document 主页面。
2. 创建一个使用状态栏和工具栏的程序。在工具栏上创建几个按钮。当鼠标指针移到工具栏中的按钮上时，状态栏上应该显示出一些关于它的简短文本信息（为此，你需要 KDE Library Reference）。
3. 扩展第一个练习中的程序，使用户能够选择一些文本并点击按钮将所选择的文本写入 stdout。

第 18 学时　编写 KDE 应用程序的第 2 课

这一学时继续讨论怎样使用 KDE 库开发 GUI 程序。它为你介绍新的 KDE 开发知识，并说明不同 KDE 库所提供的内容。

与 Qt 库比较，KDE 库被分为几个独立的库（在第 17 学时“编写 KDE 应用程序：第一课”中所学到的）。每个这样的库都具有某种类型的类，如处理文件类和处理图形类。你已经学习了这些库中一种——libkhtmlw。在上一学时，你也看到了 libkdecore 和 libkdeui 中的几个类，即 KApplication 和 KTMainWindow。但是，libkdecore 和 libkdeui 都提供了更多的内容，这一学时将介绍这些库中的其他类（但不是所有类）。

当使用 KDE 库时，一个很有价值的资源是 KDE Library Reference，它可在 http://developer.kde.org中找到。在 KDE 开发过程中，KDE Library Reference 所具有的重要性不亚于 Qt 开发中 Qt Reference Document 的重要性。

18.1　KDE 核心库

KDE 核心库，libkdecore，包含最基本的 KDE 类，如 KApplication。在这一节，将学习一些其他类。但在这一学时中不可能介绍这个库的全部内容。你应该阅读这一节及本学时中其他各节中对各个库的介绍。但应该记住，在编译时应该与你所用类库一起链接。

18.1.1　用 KAccel 类创建键盘快捷方式

KAccel 类用于定义所谓的加速键。加速键使用户能够使用键盘访问某种功能。通常访问加速键的方法是按下 Ctrl 键再按其他键。程序清单 18-1 给出一个简单的例子。

程序清单 18-1　　用 KAccel 添加加速键

```
1: #include <kapp.h>
2: #include <ktmainwindow.h>
3: #include <kaccel.h>
4:
5: int main( int argc, char **argv )
6: {
7:         KApplication a( argc, argv );
```

```
 8:          KTMainWindow w;
 9:          w.resize( 100, 100 );
10:          a.setMainWidget( &w );
11:
12:          KAccel acc( &w );
13:          acc.insertItem( "Quit", "Quit", "CTRL+Q" );
14:          acc.connectItem( "Quit", &a, SLOT( quit() ) );
15:
16:          w.show();
17:          return a.exec();
18: }
```

第 12 行创建 KAccel 对象，acc 之后，使用 KAccel::insertItem()函数创建一个新的加速项（第 13 行）。传递给该函数的两个参数（"Quit"和"Quit"）分别表示动作的局部名称和内部名称。之后，使用内部名称将该加速键与一个槽连接。最后一个参数定义你想使用的组合键（这里为 Ctrl+Q）。在第 14 行，KAccel::connectItem()函数将新创建的加速项连接到 QApplication::quit()槽。

现在来测试这个程序。事实上，它做的并不多，你将看到窗口是空的。但是，如果你按下 Ctrl 键并保持，之后按 Q 键，程序将退出。

更多信息请参考 kaccel.h 文件、ckey.h 文件和/或库参考。实际上，KAccel 包含几个其他有用功能。例如，从文件中读取加速键设置等。

18.1.2 用 KPixmap 类管理图像

当然，这个类与 QPixmap 非常类似。但是，它向标准的 QPixmap 类增加了两个非常实用的功能：WebColor 和 LowColor。在使用 256 种颜色显示时，这两种颜色模式非常有用。

在 WebColor 模式下，所有图像总是被抖动(dither)到 Netscape 调色板。换句话说，所有应用程序能够共享 Netscape 调色板，很少几种颜色被分配使用，因此将有更多的空余色元。WebColor 为 Kpixmap 的缺省颜色模式。

在 LowColor 模式下，按照 KDE 图标调色板检查图像是否与它们的颜色表相匹配。如果不匹配，图像将被抖动到一个 3×3×3 的极小色体。将 LowColor 用于背景图像、桌面图标等等，能够保证桌面上所显示对象使用的颜色种类不超过 40 种。

为了装载 KPixmap 图像对象，你应该使用 KPixmap::load()函数。更多信息请参看 kpixmap.h 文件或库参考。

18.1.3 用 KProcess 类启动子进程

KProcess 类用于在 KDE 程序中启动一个子进程。使用 KProcess，你不必关心 UNIX/Linux 系统中所使用的相当复杂的进程系统。

KProcess 将<<操作符定义为插入命令和为这些命令插入参数。为了启动执行，应该使用适当的参数调用 KProcess::start()函数。例如，为了执行 ls -als /命令，应该执行下面代码：

```
KProcess p;
P << "ls" << "-als" << "/";
p.start( KProcess::DontCare, KProcess::Stdout );
```

这里，告诉 KProcess 对象 p 用参数-als 和/执行 ls 命令。之后，使用 KProcess::start()函

数开始执行。这个函数带 0 到 2 个参数（二者都有缺省值）。

第 1 个参数为枚举值，它代表应用程序与这个进程之间的关系,在这里定义为 DontCare，它使程序不必关心子进程是否已经完成。其余选项有 NotifyOnExit 和 Block。NotifyOnExit 为缺省值。如果使用它，当子进程结束时将发射 KProess::processExited()信号。如果选择使用 Block，则在子进程结束前，程序将被阻塞。在多数情况下，这不是你所希望的。

第 2 个参数也为枚举值。它表示应用程序与子进程之间的通信类型。在这个例子中，选择的是只要求与子进程的 stdout 通信。这里共有 6 种选择：NoCommunication、Stdin、Stdout、Stderr、AllOutput 和 All。缺省取值为 NoCommunication。

如果选择某些通信类型，通过连接 KProcess::receivedStdout()和 KProcess::receivedStderr()信号，能够从 stdout 和 stderr 获得信息。也可以使用 KProcess::writeStdin()向子进程的 stdin 写入数据。更多信息请参看库参考或 kprocess.h 文件。

应该浏览一下 KShellProcess 类，它与 KProcess 有关。KShellProcess 类用于通过 UNIX/Linux 外壳执行子进程。

18.1.4　通过 KWM 类与 Window Manager 交互

KWM 类包含很多与 Window Manager 的交互函数。表 18-1 列出一些常用的 KWM 函数。很多函数在 K Window Manager（KDE 所使用的 Window Manager）以外的窗口管理器下也能很好地工作，但是，建议你使用 K Window Manager，以获得最佳效果。大多函数需要指定所使用的窗口，这通过将代表窗口的窗口对象做参数传递给函数来实现。

表 18-1　KWM 成员函数

函　数	描　述
KWM::setIcon()	使用这个函数设置窗口最小化时所显示的位图
KWM::setDecoration()	使用这个函数设置窗口装饰，共有 3 种选择： noDecoration：不装饰窗口 normalDecoration：正常装饰 tinyDecoration：使窗口成为一个小帧
KWM::currentDesktop()	这个函数返回当前可见桌面号。如果你想使程序在不同的桌面下运行时具有不同的功能，这个函数就很有用
KWM::switchToDesktop()	这里，将需要切换到的可见桌面号做参数传递给这个函数
KWM::numberOfDesktops()	返回可见桌面数量
KWM::setNumberOfDesktops()	设置可见桌面数量
KWM::activeWindow()	这个函数返回当前活动窗口
KWM::setWindowRegion()	这个函数带两个参数：一个整数，表示你所指的是哪个可见桌面，和一个 QRect 对象，它表示窗口显示所在矩形区域。如果你想使屏幕上的一些静态对象即使在其他窗口最大化时也是可见的，这个函数就非常有用
KWM::geometry()	返回一个 QRect 对象，它表示参数所指定窗口的几何尺寸
KWM::geometryRestore()	将一个最大化的窗口恢复到其原始大小
KWM::isActive()	如果参数所指定窗口为当前焦点则返回 TRUE
KWM::moveToDesktop()	将一个窗口移动到另一个可见桌面
KWM::setGeometry()	用一个 QRect 对象设置窗口几何尺寸
KWM::setGeometryRestore()	（用一个 QRect 对象）设置窗口的原始尺寸
KWM::move()	将窗口移动到 QPoint 对象定义的某个位置

续表

函　数	描　述
KWM::setMaximize()	最大化或将窗口大小设置为其原始大小。这个函数带两个参数：一个窗口对象和一个布尔值
KWM::setIconify()	和前一个函数具有相同的参数，但是，这个函数用于判断窗口是否应该为图标状
KWM::close()	用于关闭窗口

作为用户，使用 K Window Manager 所能做到的程序员通过 KWM 类也能实现。当使用 KWM 类时，表 18-1 是一个很好的起点。但是，你还需要查阅 KWM 类文档以进一步了解其他功能函数。

18.2　KDE 用户接口库

KDE 用户接口库，libkdeui，具有所有用户接口类，包括创建按钮和标签类。在第 17 学时，你已经学习了 libkdeui 中最常用的类，KTMainWindow，和其他一些类。但是，libkdeui 还提供了许多其他类。实际上，libkdeui 所包含的类太多，在这一节中不可能完全介绍它们。表 18-2 列出一些常用类。它使你初步了解哪些类可以使用，你可以通过库参考查阅到更多关于这些类的信息。

表 18-2　　KDE 用户接口库中的常用类

函　数	描　述
KButtonBox	这是一个按钮容器类，用于在水平方向或垂直方向放置按钮
KColorButton	这个部件用于让用户选择颜色
KContainerLayout	这是一个用于布置部件的 KDE 类
KDatePicker	KDatePicker 类用于创建日期提取部件
KEdit	KEdit 类基于 QMultiLineEdit 类。它预定义有文件打开和保存等功能。使用这个部件能够很快编写出一个全功能的文本编辑器
KFontDialog	KDE 中的这个类与 QFontDialog 功能相同
KIconLoaderButton	这个类创建图标装载按钮。点击这个按钮将打开一个图标对话框（由 KIconLoaderDialog 创建）
KLed	创建一个发光二极管，它在实际中非常有用。例如，如果在程序中指示某些事情打开或关闭状态
KListSpinBox	使用户循环提取一个定义的项目列表
KMsgBox	这是一个增强的 QMessageBox 类。它所具有的按钮数可多达 4 个，并可具有国际化按钮文本
KNoteBook	提供一个选项卡对话框，与 QTabDialog 相比，它提供许多新的功能
KNumericSpinBox	使用户能够循环提取一定范围内的数值
KPanner	一个简单部件，它使用户能够控制两个部件大小
KProgress	如果你想使用 KDE 类而不是 Qt 类，则使用这个类代替 QProgressBar 和 QProgressDialog
KQuickHelpWindow	使用这个类创建一个具有快速帮助文本的窗口
KQuickTip	创建一个 KDE 帮助系统。在编写这本书时，这个类仍在开发之中
KRadioGroup	这个类用于在工具条中创建单选按钮组
KSeparator	用于创建一条水平线。使用这个类可以确保所有 KDE 程序中的水平线看起来完全相同

续表

函　数	描　述
KSlider	这个类与 QSlider 基本相同。但是，建议你使用这个类，以使 KDE 程序中的所有滑动条标准化
KStatusBar	创建状态栏部件
KTabBar	KTabBar 与 QTabBar 非常类似。唯一的差别是当部件不能放置在选项卡中时 KTabBar 提供滚动按钮
KTabCtl	与 QTabDialog 非常类似。但是，KTabCtl 不创建按钮，因此，它不局限于对话框
KTabListBox	提供一个多列列表框。用户可以调整列尺寸
KTreeList	提供一个显示结构化数据（例如，目录树）的部件
KWizard	这个类用于创建所谓的向导，类似于 Microsoft Windows 中常用的安装向导或配置向导。在向导中所填些的数据项顺序是很重要的

如前所述，libkdeui 所具有的类远比表 18-2 所列出的类要多。关于 libkdeui 的更多信息请查阅 KDE 库参考。

18.3　KDE 文件操作库

KDE 有它自己的文件操作库，叫做 libkfile。它包含处理文件和目录所需的所有功能。在这一节，你将接触 libkfile 中的类，并学习怎样使用它们。

18.3.1　用 KDirDialog 类选择目录

KDirDialog 类是 QFileDialog 类的增强版本。它提供一个对话框用于选择文件。使用 KDirDialog 类非常简单。程序清单 18-2 给出一个例子：

程序清单 18-2　　用 KDirDialog 创建目录对话框

```
 1: #include <kapp.h>
 2: #include <ktmainwindow.h>
 3: #include <kfiledialog.h>
 4:
 5: int main( int argc, char **argv )
 6: {
 7:         KApplication a( argc, argv );
 8:         KTMainWindow w;
 9:         w.resize( 100, 100 );
10:         a.setMainWidget( &w );
11:
12:         QString dir = KDirDialog::getDirectory( "file:/" );
13:
14:         w.show();
15:         return a.exec();
16: }
```

这个程序调用 KDirDialog::getDirectory()函数（第 12 行），它在屏幕上显示出目录对话框，因此用户能够选择目录。当用户完成并点击 OK 按钮时，目录名称被存储到 dir 中（也在第 12 行处理）。在第 12 行还设置 KDirDialog 对象的起点，这里被设置到 file:/。



与 Windows 98 一样，KDE 依赖于 URL 而不是目录。因此，在程序清单 18-2 中 KDirDialog 对象的起点被设置为 file:/，而不是/。

调用下面命令编译这个程序：

```
# g++ -I$KDEDIR/include -L$KDEDIR/lib -lqt -lkdecore -lkdeui -lkfile -lkfm
  18lst02.cpp -o Listing-18.2
```

注意，需要链接 libkfile 和 libkfm 库。图 18-1 为该程序运行结果。

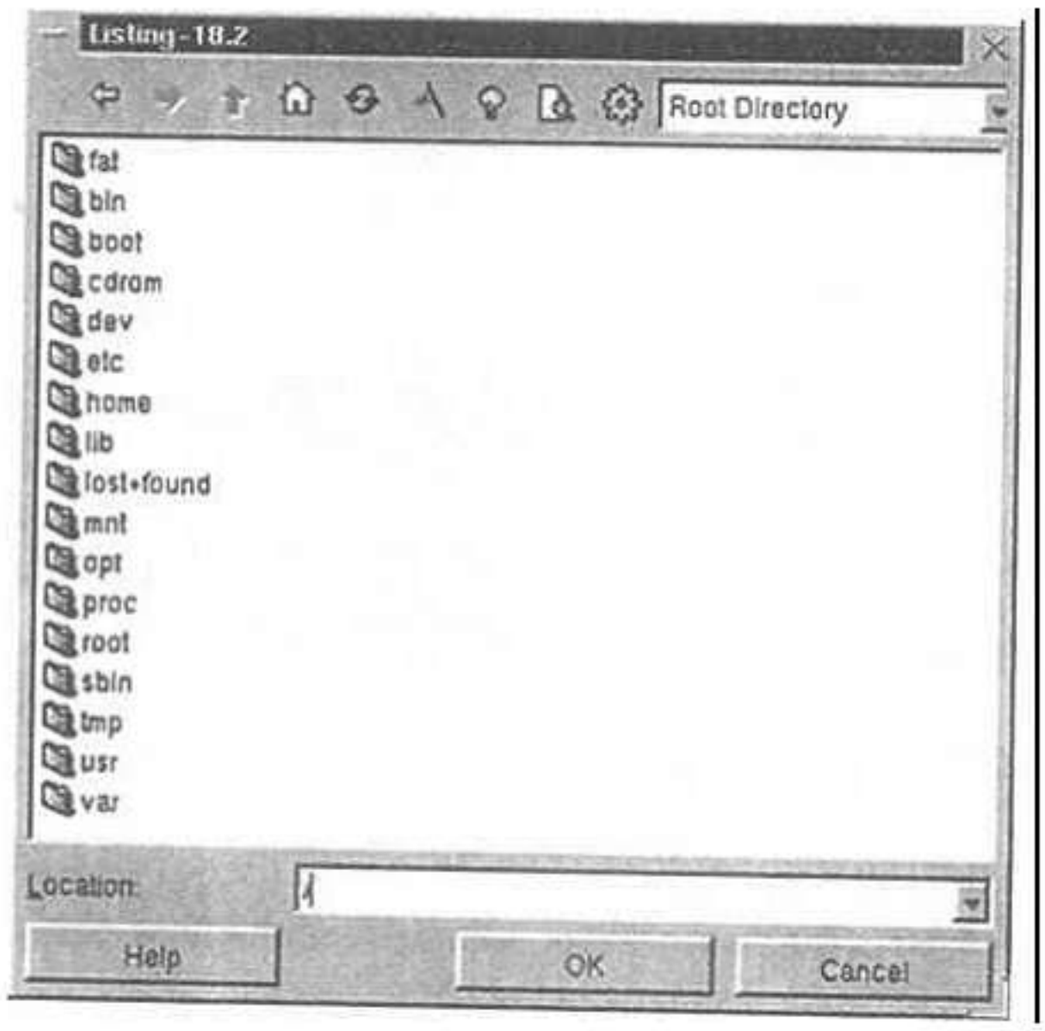


图 18-1 KDirDialog 类创建的目录对话框

18.3.2 用 KFileDialog 类选择文件

KDirDialog 类所创建的对话框与图 18-1 类似。但是，它用于选择文件，而不是目录。看下面一行代码：

```
QString dir = KFileDialog::getOpenFileName( "file:/ ", "*" );
```

这将显示文件对话框，让用户选择一个文件，之后将文件名存储在 dir 中。传递给 KFileDialog::getOpenFileName()的两个参数表示 KFileDialog 从目录树中的哪里地方开始查找文件，以及使用哪个过滤器。这里，过滤器被设置为*，这使文件对话框显示出所有文件。如果只想看到以.txt 结尾的文件，则应该将过滤器修改为*.txt。

18.3.3 用 KFileInfo 类读取文件信息

与 QFileInfo 类似，KFileInfo 用于读取各种文件信息。看下面几行代码：

```
#include <qfilesimpleview.h>

KFileInfo file( "/vmlinuz" );
```

```
cout << "The file's size is: " << file.size( ) << endl;
cout << "The file belongs to the group " << file.group( ) << endl;
cout << "The file's owner is " << file.owner( ) << endl;
cout << "The access permission for the file is " << file.access( ) << endl;
```

在程序中实现这些代码，你将在 stdout 上得到一些输出信息。这些信息与下面内容类似：

```
The file's size is: 444199
The file belongs to the group root
The file's owner is root
The access permission for the file is -rw-r-r--
```

这非常简单，不是吗？实际上，在很多情况下 KFileInfo 类比 QFileInfo 类更容易使用。关于这个类的更多信息，请查阅库参考中的 libkfile 类。

18.3.4　用 KFilePreviewDialog 预览文件

文件预览是一个很有用的功能，它使用户在打开文件之前能够看到文件中的一少部分内容。KDE 提供的 KFilePreviewDialog 类实现这一功能。与使用所有其他文件对话框一样，只用单行代码就能够实现它：

```
QString file = KFilePreviewDialog::getOpenFileName( );
```

这一行将显示文件预览对话框，如图 18-2 所示。

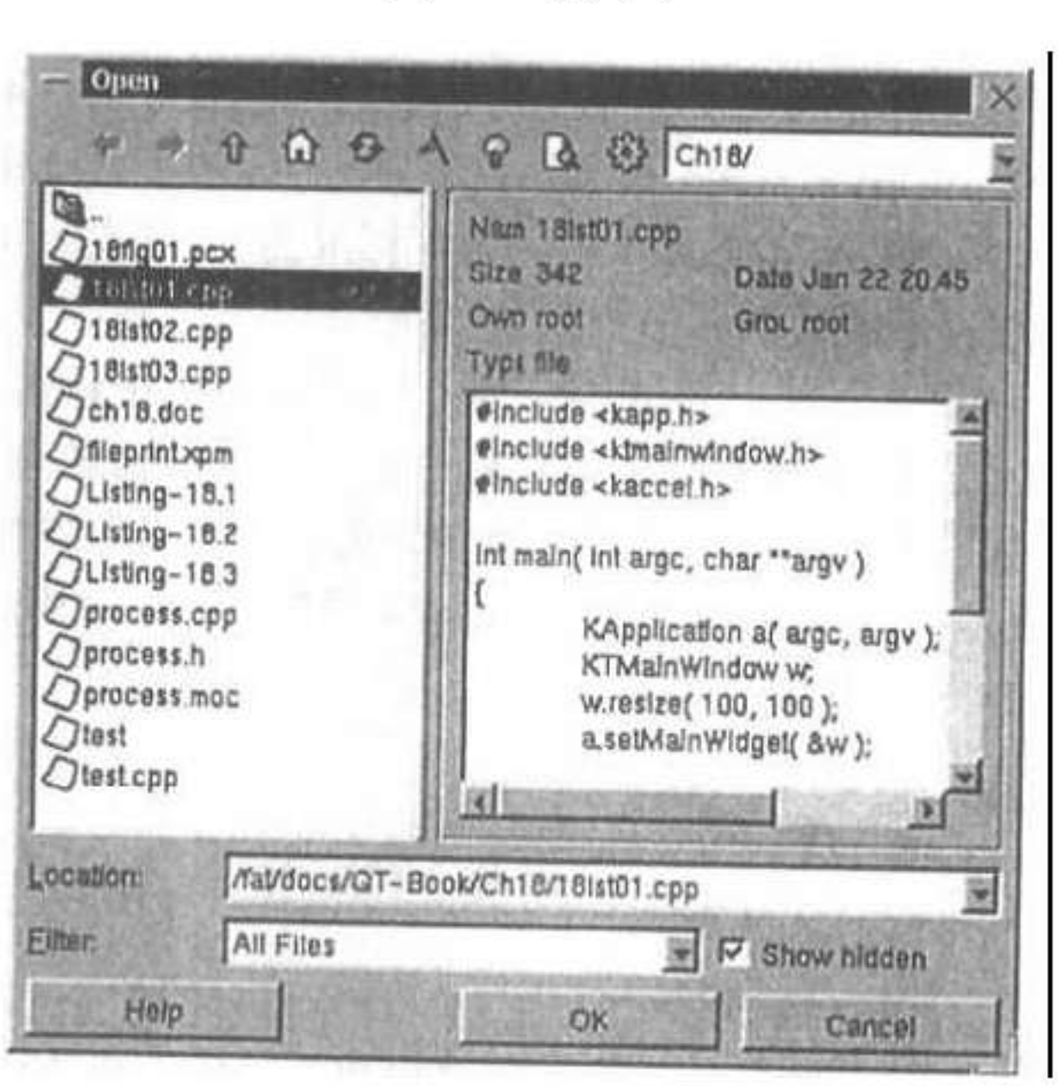


图 18-2　KFilePreviewDialog 类创建的文件预览对话框

在编写本书时，只能预览文本文件和位图文件。但是，如果需要预览一个不支持的文件格式，你自己完全可以实现对那种文件格式的支持，尽管它可能是一个相当复杂的项目。

如你所看到的，在处理文件时，KDE 提供几个很有用的对话框。但是，对于实际的文件 I/O 操作，你必须使用标准的 Qt 类。

18.4 其余 KDE 库

你已经看到了 libkdecore、libkdeui、libkdehtmlw 和 libkfile。但是，还需要了解其他几个小库。

首先，KDE 还带有一个叫做 libkimgio 的库。这个库带有支持各种图像 I/O 操作的 KDE 类。在编写本书时，libkimgio 未包含太多有用内容。但是，它仍在进一步开发之中，将来肯定会有令人感兴趣的内容。

libkspell 是 KDE 库家族中的又一个成员。在编写本书时，libkspell 所包含的唯一一个类为 KSpell，它是国际拼写检查程序 ISpell 的程序设计接口。

最后一个库，libkab，具有访问 KDE 地址薄功能。



注意，这些是 KDE 1.1.2 中所包含的库。在 KDE 2.0 中将包含更多库。

18.5 小 结

如在第 17 学时中所提到的，你不必只是因为在编写 KDE 程序而使用 KDE 类。如果在 KDE 类中不能找到任何能够使程序更加优化的函数，则没有理由使用它。只有当你感觉它们能够使程序更加优化，或者是 KDE 程序必须遵守一种标准时，才需要使用 KDE 类。只有当使用 KApplication 和 KTMainWindow，并实现标准的 KDE 帮助菜单时，你才有权利将你的程序称做 KDE 程序（多数人的看法）。

但是，随着 KDE 项目的开发和 KDE 类的进一步优化，你应该使用 KDE 类所提供的强大功能。一个很好的例子是 KFilePreviewDialog 类。只需要修改程序中的单行代码，用 KFilePreviewDialog 类代替 QFileDialog，就能够使你的程序变得更加容易使用。

需要记住的另一点是 KDE 2.0 计划在 2000 年 4 季度发行。对用户和开发人员来说，这都是一次重大更新，它将包含很多新的功能。

18.6 问题与答案

问：当我编译 KDE 程序时，出现符号无法解析错误。这是为什么？

答：你忘了链接程序所需要的库。因此，要保证这一点是正确的。也要注意，当你使用 libkfile 库时，还需要链接 libkwm。

问：我有一些关于 KWM 类的问题，不是所有函数都正常使用。这是为什么？

答：可能你所使用的 Window Manager 不是 K Window Manager。这种 Window Manager 不能理解 KWM 类所发出的指令。你应该使用 K Window Manager。

18.7　作　业

完成下面的问题和练习将有助于你牢记这一学时中所学内容。回答这些问题能够确保你理解这学时所教的关于 KDE 程序设计知识。

18.7.1　测验

1. KDE 核心库中有哪些内容？
2. 如果你想在程序中启动一个子进程，应该使用哪个类？
3. 如果子进程正好是一个外壳命令，你应该使用哪个类？
4. 与 QPixmap 相比，KPixmap 具有哪些优点？
5. 在 KDE 程序中能够将一个窗口最小化吗？
6. libkdeui 中包含哪种类？
7. 当编写 KDE 应用程序时，是否总需要使用 libkdeui 中的类构造用户界面？
8. 当需要执行一些文件 I/O 操作时，你是否应该使用 libkfile 中的类？

18.7.2　练习

1. 编写一个 KDE 程序，它使用 KProcess 类执行一个子进程，之后将这个进程的输出插入到一个 KEdit 对象中。
2. 使用 KFontDialog 使用户选择字体。之后，用用户所选择的字体创建一个标签。
3. 创建一个 KTabBar 对象。向其中插入一些它不能容下的部件。将会发生什么事情？
4. 编写一个程序，其中创建几个 KFileInfo 对象（所有的对象都应该参照不同的文件）。之后创建一个函数将这些文件按大小排序。将这些文件名称插入到一个 KEdit 对象中，一行一个，最大的文件首先插入。你可能还想显示其他一些文件信息，如所有者，组等等。

第 19 学时　使用 Qt 的 OpenGL 类

OpenGL 是一套用于绘制 3D 图形的标准 API。它与平台无关，并因此被用于很多不同的系统。缺省时，Windows NT 和一些商用 UNIX 系统都带有 OpenGL 库。但是，在其他一些系统上需要手工安装它们。幸运的是，有一套免费的 OpenGL，叫做 MESA，可用于 UNIX/Linux。这是这一学时中将使用的 OpenGL 实现。尽管 MESA 通常不包含 OpenGL 的最新功能，但是它很好用，并且是免费的！也可以使用其他厂家的 OpenGL 实现，如 SGI OpenGL。

Qt 所包含的几个类使得在 Qt 程序中能够更加容易地集成 OpenGL 功能。这样做，你将收益很多，因为可以使用 Qt 为 OpenGL 程序创建 GUI，并且不需要自己去重新创建按钮、菜单、工具栏等等。

在这一学时，将学习怎样使用 Qt 开始 OpenGL 程序设计。但是，这里并未讨论所有的 OpenGL 功能和元素——这些材料足以写另外一本书。为了讨论 OpenGL 和三维图形，你应该访问www.opengl.org站点，那里有很多关于 OpenGL 的文档。在学习这一学时之前，你至少应该有一些基本的 OpenGL 知识。

19.1　建立 OpenGL 开发环境

在开始之前，需要完成几项工作。首先，应该获取和安装 MESA（如果你还没有任何 OpenGL 实现的话）。之后，需要编译 Qt 的 OpenGL 扩展。

19.1.1　获取和安装 MESA

从www.mesa3d.org站点总能查找到最新的 MESA 版本。在编写本书时，其最新版本为 3.1。最近可能会有变化，因此，你要确保得到的是最新版本。其大小在 1.5～2.0MB 之间。



你的 UNIX/Linux 分发程序中可能带有 MESA。如果是这样，就不需要去编译和安装源分发程序，而是可以直接使用分发程序所带的二进制版本。

当把文件下载到硬盘时，需要解开文件：

```
# tar xvfz MesaLib-3.1.tar.gz
```

这在硬盘上创建一个 Mesa-3.1 的目录。切换到这个目录：

```
# cd Mesa-3.1
```

之后，启动它所提供的 configure 实用程序：

```
# ./configure
```

configure 程序将检查系统，并使 MESA 在编译时能够适应你所使用的系统。运行 make 程序开始编译：

```
# make
```

现在，编译开始，你将看到许多信息在屏幕上滚动。编译需要一定的时间，如果你的系统较慢时间则会更长，因此，要耐心一点。当编译结束时，即可安装该库：

```
# make install
```

这将 MESA 库安装到/usr/local/lib，并将头文件安装到/usr/local/include/GL。



如果要将 MESA 安装到其他地方，你可以在 configure 实用程序中使用--prefix 选项。例如，如果希望将它安装在/usr/lib 和/usr/include/GL 下，你需要按下面格式执行 configure:

```
# ./configure --prefix=/usr
```

现在，MESA 库已经安装好，你可以开始编译 Qt 的 OpenGL 扩展。

19.1.2 编译 Qt 的 OpenGL 扩展

Qt 的 OpenGL 扩展位于$QTDIR/extensions/opengl/src 下。为了编译和安装它，需要运行这个目录中的 make：

```
# make
```

这将编译库，并将它安装在$QTDIR/lib 下。这个库为静态类型，其名称为 libqgl.a。至此，所有设置都已经完成。

19.2 Qt 的 OpenGL 类

Qt 的 OpenGL 扩展由 3 个类组成：QGLWidget、QGLContext 和 QGLFormat。这些类使你在使用 Qt 部件创建用户界面的同时能够使用 OpenGL 功能创建图形。这一节讨论这 3 个类，使你更好理解怎样和什么时候应该去使用它们。

19.2.1 QGLWidget——OpenGL 部件

QGLWidget 类是 Qt OpenGL 扩展中最重要的一部分。在 QGLWidget 的一个子类中，能够初始化和创建 OpenGL 图形。你应该将 OpenGL 代码插入到 3 个静态函数 QGLWidget::initializeGL()、QGLWidget::paintGL()和 QGLWidget::resizeGL()中。下面将更深入地介绍这些函数。

一、理解 initializeGL()函数

在你的 QGLWidget::initializeGL()函数实现里，应该添加对 OpenGL 子系统的初始化代码。这通常包含几个 OpenGL 函数调用，由它们设置 OpenGL 图形颜色和形状。有时，这些函数也包含对 QGLWidget 成员函数的调用。这里是一个例子：

```
void GLBox::initializeGL( )
{
    qglClearColor( black );
    glShadeModel( GL_FLAT );
}
```

首先，调用 Qt 函数 qglClearColor()（所有以 qgl 开头的函数均是 Qt OpenGL 实现的一部分，而不是实际的 OpenGL API）。这个函数实际上去调用 OpenGL 函数 glClearColor()，它设置清除颜色（基本上为背景颜色）。尽管这里也能够使用 glClearColor()，但是，qglClearColor()函数更加容易使用，因为它能够处理 QColor 对象。

此后，调用 OpenGL 函数 glShadeModel()。使用这个函数能够设置底纹。在这个例子中，将底纹设置为无以提高性能。

注意，应该将所有 OpenGL 相关的初始化代码放置在 initializeGL()函数中，而不是在类构造函数中。

二、理解 paintGL()函数

QGLWidget::paintGL()是基于 Qt 的 OpenGL 程序中最令人感兴趣的函数。这里，你可以放置实际的 OpenGL 绘图代码。这个函数具有与 paintEvent()函数相同的功能，尽管 paintGL()应该使用 OpenGL 而不是 QPaint 进行绘图。

当然，你在这个函数中实际写入的代码依赖于需要绘制的图形。因为应该使用 OpenGL 绘图，所以需要熟悉 OpenGL 绘图函数才能实现 paintGL()。它通常由各种各样的 OpenGL 绘图函数和 glEnd()函数调用组成，glEnd()函数标识某个绘图操作的结束。

在本学时后面的程序清单 19-1 中，你将看到一个 paintGL()函数的例子。

三、理解 resizeGL()函数

每当调整部件大小时就调用 QGLWidget::resizeGL()函数。其用途与 QWidget::resizeEvent()函数类似，应该使用它修改与尺寸相关的参数——例如，对象的大小和位置，以及窗口的大小等。

在调用 QGLWidget::resizeGL()函数时，总是使用两个整数做参数。第 1 个参数表示部件的新宽度，第 2 个参数表示部件的新高度。

19.2.2　QGLContext——绘制 OpenGL 图形

如 Qt Reference Document 所述，“QGLContext 类封装一个 OpenGL 绘图环境”。那么，其意思是什么呢？OpenGL 环境是一个用于输出图形的对象。它通常是窗口的一部分，但也可将它设置为位图之类的东西。

QGLWidget 自动创建一个绘图环境（一个 QGLContext 对象），因此，如果不需要改变这个环境，就不必关心它。但是，如果要将一个 OpenGL 图形输出到一个位图。你可以创建一个 QPixmap 对象，之后再为它创建一个 QGLContext 对象。例如：

```
QPixmap pixmap;
QGLContext *pcx = new QGLContext( QGLFormat( ), &pixmap );
```

如你所看到的，QGLContext 构造函数带两个参数。第 1 个参数是一个 QGLFormat 对象（关于 QGLFormat 类请阅读下一节），第 2 个参数是一个 QPaintDevice 对象。因此，能够将一个 OpenGL 图形绘制到所有继承 QPaintDevice 的 Qt 类。之后，通过改变 QGLWidget 对象的绘图环境和调用 QGLWidget::updateGL()，将 OpenGL 图形写入位图。例如：

```
MyQGLWidgetObject -> setContext( pcx);
MyQGLWidgetObject -> updateGL( );
```

注意，这里省略了 QWidget::setContext()函数的两个参数，因为它们都有缺省取值。关于这个函数及其参数的更多信息请参看 Reference Document。

19.2.3 QGLFormat——设置环境显示格式

如果不满足于 OpenGL 绘图环境的缺省设置，可以使用 QGLFormat 类设置自己的选项。但是，为了使用这个类，你需要很好理解 OpenGL 的更高级功能，例如，字母混合和模板填充等。为了对 QGLWidget 对象改变 QGLFormat 对象，你应该使用 QGLWidget::setFormat()函数。用 QGLFormat::format()函数可以检索当前 QGLFormat 对象。为环境设置用户 QGLFormat 对象的另外一种方法是将它作为 QGLContext 构造函数的第一个参数。

在创建 QGLFormat 对象之后，可以使用表 19-1 中所描述的函数来设置各种选项。所有函数均带一个 bool 值做参数。

表 19-1　　QGLFormat 成员函数

函　数	描　述
QGLFormat::setDoubleBuffer()	将这个函数的参数设置为 TRUE，则打开双缓冲。将参数设置为 FALSE，则使用单缓冲。双缓冲技术首先在屏幕外缓冲区中绘制图形，当绘图完成后，再将缓冲区内容拷贝到屏幕。这能够避免闪烁，并有助于提高性能
QGLFormat::setDepth()	使用这个函数能够打开或关闭深度缓冲区。深度缓冲区技术也称做 z-缓冲。这一技术赋于像素一个 z-值，它表示像素和观察者之间的距离。像素的 z-值越大，它离观察者的距离越近。具有最高 z-值的像素被最先绘出
QGLFormat::setRgba()	如果将参数设置为 TRUE，这个函数打开 RGBA 颜色模式。如果参数为 FALSE，则打开颜色索引模式。RGBA 是 RGB 的扩展版本。它向颜色添加第 4 个值——α 参数
QGLFormat::setAlpha()	如果这个函数调用时参数为 TRUE，它将为帧缓冲启动 α 通道。启动 α 通道时，RGBA 中的 α 参数则说明一个像素的透明度
QGLFormat::setAccum()	当其参数为 TRUE 时，这个函数启动累积缓冲功能。这一功能用于产生模糊效果和多重暴光效果
QGLFormat::setStencil()	当其参数为 TRUE 时，这个函数启动模板缓冲。模板缓冲用于屏蔽掉在屏幕中某些部分所绘制的图形
QGLFormat::setStereo()	用 TRUE 参数调用这个函数时将打开立体缓冲。立体缓冲技术用于提供额外的颜色缓冲，以产生左眼和右眼图像
QGLFormat::setDirectRendering()	用 TRUE 参数调用这个函数时，OpenGL 将直接从硬件到屏幕绘图，而越过窗口系统
QGLFormat::setOverlay()	使用这个函数启动覆盖平面。这将在覆盖平面中创建一个额外的环境

你可能已经注意到，当创建 3D 图形时，需要了解许多术语。但是，所有这些都可从www.opengl.org站点中各种各样的 OpenGL 文档中找到解释。

注意，QGLFormat 也带有一些函数用于检索所有这些选项，因此，你能够判断一个选项当前是否启动。

19.3　编写、编译和运行基于 Qt 的 OpenGL 程序

在这一节，你将看到一个 Qt 与 OpenGL 集成的例子。这个例子用 OpenGL 创建一个窗口，它具有 1 个 OpenGL 绘图区域和 3 个用 Qt 的 Qslider 类创建的滑动框。通过使用 Qt 简单的信号和槽功能，能够使 OpenGL 图形随着滑动框的拖动而改变。

下面将先阅读一下代码。尽管例子中的代码已经给出了很好的注释，但是需要你具有一些基本的 OpenGL 程序设计知识才能掌握它。之后，学习怎样编译和运行这个例子，以及需要链接哪个库，等等。

19.3.1　阅读代码

如前所述，如果不具备 OpenGL 程序设计知识，你将不能理解和掌握这个例子。三维图形设计有很多特殊的术语和复杂技术。因此，你应该先阅读www.opengl.org中的一本好教程，之后就能够很好理解这个例子。

程序清单 19-1 相当长，因此你应该将它分为几个独立的文件。至少需要将 GLBox 类声明放入一个文件中，以便能够使用元对象编译器。

程序清单 19-1　　怎样连接 OpenGL 和 Qt

```
/***** Start glbox.h *****/
//Since the class includes custom
//slots, remember to use MOC on it
//and include the output in glbox.cpp
#include <qgl.h>

class GLBox : public QGLWidget
{
Q_OBJECT

public:
        GLBox( QWidget* parent );
        ~GLBox();

public slots:
        void setXRotation( int degrees );
        void setYRotation( int degrees );
        void setZRotation( int degrees );

protected:
        void initializeGL();
        void paintGL();
        void resizeGL( int w, int h );
```

```
24:         virtual GLuint makeObject();
25:
26: private:
27:         GLuint object;
28:         GLfloat xRot, yRot, zRot, scale;
29: };
30: /***** End glbox.h *****/
31:
32: /**** Start glbox.cpp *****/
33: #include <qgl.h>
34: #include "glbox.moc"
35:
36: //In the constructor, we only set some values.
37: //We don't do any OpenGL function calls here:
38: GLBox::GLBox( QWidget *parent ) : QGLWidget( parent )
39: {
40:     //Set the default rotation:
41:     xRot = yRot = zRot = 0.0;
42:     //Set the default scale:
43:     scale = 1.25;
44:     //Set our OpenGL object to 0:
45:     object = 0;
46: }
47:
48: //Delete the display list:
49: GLBox::-GLBox()
50: {
51:     //The glDeleteLists() function is
52:     //used to delete the display list.
53:     //The second argument represent the
54:     //number of display lists to be deletec
55:     glDeleteLists( object, 1 );
56: }
57:
58: //Paint the box using the OpenGL functions
59: //provided by MESA:
60: void GLBox::paintGL()
61: {
62:     //Clear the buffer and enable
63:     //color drawing:
64:     glClear( GL_COLOR_BUFFER_BIT );
65:
66:     //Replace the current matrix with
67:     //the identity matrix:
68:     glLoadIdentity();
69:
70:     //Translate the current matrix with
71:     //the following x, y, and z values:
72:     glTranslatef( 0.0, 0.0, -10.0 );
73:
74:     //Scale the current matrix with
75:     //the following x, y, and z values.
76:     //Use the predefined scale-value
77:     //for all axes (1.25):
78:     glScalef( scale, scale, scale );
79:
```

```
80:     //Rotate the matrix around its
81:     //x-axis by the angle xRot. The
82:     //value of xRot is changed as you
83:     //drag the top slider:
84:     glRotatef( xRot, 1.0, 0.0, 0.0 );
85:
86:     //Rotate the matrix around its
87:     //y-axis by the angle yRot. The
88:     //value of yRot is changed as you
89:     //drag the middle slider:
90:     glRotatef( yRot, 0.0, 1.0, 0.0 );
91:
92:     //Rotate the matrix around its
93:     //z-axis by the angle zRot. The
94:     //value of zRot is changed as you
95:     //drag the bottom slider:
96:     glRotatef( zRot, 0.0, 0.0, 1.0 );
97:
98:     //Executes a display list. This
99:     //function call will draw the new
100:    //box to screen:
101:    glCallList( object );
102: }
103:
104: //Initialize the OpenGL sub-system:
105: void GLBox::initializeGL()
106: {
107:    //Use black as background color:
108:    qglClearColor( black );
109:
110:    //Generate an OpenGL display list.
111:    //A display list is a group of OpenGL
112:    //commands. On our case, these commands
113:    //are defined in the makeObject() function.
114:    //object is the integer name of the display list:
115:    object = makeObject();
116:
117:    //Set the shade model to flat.
118:    //This is usually done to increase
119:    //performance:
120:    glShadeModel( GL_FLAT );
121: }
122:
123: //This function will be called when the
124: //widget is resized. We access its new
125: //width and height through the variables
126: // w and h:
127: void GLBox::resizeGL( int w, int h )
128: {
129:    //Set the viewport. The first two
130:    //arguments (0, 0) represent the
131:    //lower left corner of the viewport.
132:    //The last two arguments represent
133:    //the width and height of the viewport.
134:    glViewport( 0, 0, (GLint)w, (GLint)h );
135:
```

```
136:     //This function sets the current matrix.
137:     //In this case, we set the current matrix
138:     //to the projection matrix.
139:     glMatrixMode( GL_PROJECTION );
140:
141:     //Replace the current matrix with the
142:     //identity matrix:
143:     glLoadIdentity();
144:
145:     //This function sets perspective of the matrix.
146:     //See the documentation found on www.opengl.org
147:     //for more information:
148:     glFrustum( -1.0, 1.0, -1.0, 1.0, 5.0, 15.0 );
149:
150:     //Set the current matrix to the modelview matrix:
151:     glMatrixMode( GL_MODELVIEW );
152: }
153:
154: //This function creates our OpenGL display list:
155: GLuint GLBox::makeObject()
156: {
157:     //Create a GLuint object that will represent
158:     //the display list:
159:     GLuint list;
160:
161:     //Create one empty display list and assign it
162:     //to list:
163:     list = glGenLists( 1 );
164:
165:     //This function starts the definition of the
166:     //new display list:
167:     glNewList( list, GL_COMPILE );
168:
169:     //Set the drawing color to white:
170:     qglColor( white );
171:
172:     //Set the width of the lines that
173:     //will be drawn:
174:     glLineWidth( 2.0 );
175:
176:     //Make the vertices we are about to
177:     //draw connected to each other:
178:     glBegin( GL_LINE_LOOP );
179:     //Draw three vertices. The three
180:     //arguments represent the x, y, and
181:     //z values of the vertices:
182:     glVertex3f(  1.0,  0.5, -0.4 );
183:     glVertex3f(  1.0, -0.5, -0.4 );
184:     glVertex3f( -1.0, -0.5, -0.4 );
185:     glVertex3f( -1.0,  0.5, -0.4 );
186:     //End the drawing:
187:     glEnd();
188:
189:     //Same as previous:
190:     glBegin( GL_LINE_LOOP );
191:     glVertex3f(  1.0,  0.5, 0.4 );
192:     glVertex3f(  1.0, -0.5, 0.4 );
```

```
193:     glVertex3f( -1.0, -0.5, 0.4 );
194:     glVertex3f( -1.0,  0.5, 0.4 );
195:     glEnd();
196:
197:     //Now, treat each pair of vertices as
198:     //an independent line segment:
199:     glBegin( GL_LINES );
200:     //First pair:
201:     glVertex3f(  1.0,  0.5, -0.4 );
202:     glVertex3f(  1.0,  0.5, 0.4 );
203:     //Second pair:
204:     glVertex3f(  1.0, -0.5, -0.4 );
205:     glVertex3f(  1.0, -0.5, 0.4 );
206:     //Third pair:
207:     glVertex3f( -1.0, -0.5, -0.4 );
208:     glVertex3f( -1.0, -0.5, 0.4 );
209:     //Fourth pair:
210:     glVertex3f( -1.0,  0.5, -0.4 );
211:     glVertex3f( -1.0,  0.5, 0.4 );
212:     glEnd();
213:
214:     //End the display list definition:
215:     glEndList();
216:
217:     //Return the list so that we can
218:     //assign it to another GLuint object:
219:     return list;
220: }
221:
222: //The following three slots will be called when
223: //its corresponding slider is dragged:
224:
225: //Set the objects rotation angle around the X-axis:
226: void GLBox::setXRotation( int degrees )
227: {
228:     //Update xRot to the new value that
229:     //is set by the top slider:
230:     xRot = (GLfloat)(degrees % 360);
231:     //Update the drawing:
232:     updateGL();
233: }
234:
235: //Set the objects rotation angle around the Y-axis:
236: void GLBox::setYRotation( int degrees )
237: {
238:     //Update yRot to the new value that
239:     //is set by the middle slider:
240:     yRot = (GLfloat)(degrees % 360);
241:     //Update the drawing:
242:     updateGL();
243: }
244:
245: //Set the objects rotation angle around the Z-axis:
246: void GLBox::setZRotation( int degrees )
247: {
248:     //Update zRot to the new value that
249:     //is the by the bottom slider:
```

```
250:     zRot = (GLfloat)(degrees % 360);
251:     //Update the drawing:
252:     updateGL();
253: }
254: /***** End glbox.cpp *****/
255:
256: /***** Start glwindow.h *****/
257: class GLWindow : public QWidget
258: {
259: public:
260:         GLWindow( QWidget *parent = 0 );
261:
262: };
263: /***** End glwindow.h *****/
264:
265: /***** Start glwindow.cpp *****/
266: #include <qwidget.h>
267: #include <qpushbutton.h>
268: #include <qslider.h>
269: #include <qlayout.h>
270: #include <qapplication.h>
271: #include <qkeycode.h>
272:
273: #include "glbox.h"
274: #include "glwindow.h"
275:
276: GLWindow::GLWindow( QWidget *parent ) : QWidget( parent )
277: {
278:     resize( 400, 300 );
279:
280:     //Create our OpenGL widget:
281:     GLBox *glbox = new GLBox( this );
282:     glbox->setGeometry( 50, 10, 340, 280 );
283:
284:     //Create a slider for controlling the X-axis:
285:     QSlider *x = new QSlider ( 0, 360, 60, 0, QSlider::Vertical, this );
286:     x->setGeometry( 10, 30, 30, 75 );
287:     x->setTickmarks( QSlider::Left );
288:
289:     //Create a slider for controlling the Y-axis:
290:     QSlider *y = new QSlider ( 0, 360, 60, 0, QSlider::Vertical, this );
291:     y->setGeometry( 10, 115, 30, 75 );
292:     y->setTickmarks( QSlider::Left );
293:
294:     //Create a slider for controlling the Z-axis:
295:     QSlider *z = new QSlider ( 0, 360, 60, 0, QSlider::Vertical, this );
296:     z->setGeometry( 10, 200, 30, 75 );
297:     z->setTickmarks( QSlider::Left );
298:
299:     //Connect the slider to its appropriate slot on our GLBox object:
300:     connect( x, SIGNAL( valueChanged(int) ), glbox,
301:              SLOT( setXRotation(int) ) );
302:     connect( y, SIGNAL( valueChanged(int) ), glbox,
303:              SLOT( setYRotation(int) ) );
304:     connect( z, SIGNAL( valueChanged(int) ), glbox,
305:              SLOT( setZRotation(int) ) );
306: }
```

```
307: /***** End glwindow.cpp *****/
308:
309: /***** Start main.cpp *****/
310: #include <qapplication.h>
311: #include <qgl.h>
312:
313: #include "glwindow.h"
314:
315: int main( int argc, char **argv )
316: {
317:         QApplication a(argc,argv);
318:         GLWindow w;
319:         a.setMainWidget( &w );
320:         w.show();
321:         return a.exec();
322: }
323: /***** End main.cpp *****/
```

尽管程序清单中的每个功能都有注释，但对整个程序有一个总的了解是困难的。它只是一些非常简单的概念。

GLBox 类（在第 7 行到第 29 行中声明）负责实际的 OpenGL 绘图。它定义 3 个静态方法 paintGL()（第 60 行到第 102 行）、initializeGL()（第 105 行到第 121 行）和 resizeGL()（第 127 行到第 152 行）。这些函数的用法与前面所描述的相同。GLBox 还拥有一个函数 makeObject()（第 155 行到第 220 行），它创建 OpenGL 显示列表（从第 159 行开始）（一个显示列表是一组绘图操作），之后返回它（第 219 行）。GLBox 还有 3 个槽：setXRotation()（第 226 行到第 233 行）、setYRotation()（第 236 行到第 243 行）和 setZRotation()（第 246 行到第 253 行）。这些函数都被连接到一个 Qslider 对象的 valueChanged()信号。但这些在 GLWindow 的构造函数中实现（更确切地说，在第 300 行到第 305 行）。这 3 个槽为 x-、y-或 z-轴设置一个新的角度值，再调用 updateGL()绘制新的图形（第 232、242 和 252 行）。

之后，定义 GLWindow 类（在第 257 行到第 262 行定义）。这是一个常规 Qt 类，它基于 Qwidget。在 GLWindow 构造函数中（第 257 行到第 306 行）创建一个 GLBox 对象（第 281 行）。GLWindow 构造函数还创建 3 个 Qslider 对象（第 286 行到第 297 行），并将它们的 valueChanged()信号连接到 GLBox 对象的一个槽（第 300 行到第 305 行）。采用这种方法，能够将滑块的新值传递到相应的 GLBox 槽，该槽又使盒子的视角做同样的改变。

然后，在 main()函数中创建一个 GLWindow 类对象，并将它设置为主部件（第 319 行）。尽管是 GLWindow 使用 GLBox 创建 OpenGL 图形，但它与所有 Qt 部件的实现方法相同。

19.3.2　编译和运行例子

为了编译程序清单 19-1 中的程序，需要将这些代码分为不同的.cpp 文件和.h 文件（像程序清单中注释的那样命名这些文件）。之后，应该有 5 个文件——glbox.cpp、glbox.h、glwindow.cpp、glwindow.h 和 main.cpp。

现在，应该使用元对象编译器对 glbox.h 进行编译。因为 glbox.h 是唯一一个包含用户槽的类，因此，它也是唯一一个需要使用 MOC 的类，操作如下：

```
# moc glbox.h -o glbox.moc
```

注意，如果你所选择使用的输出文件名不是 glbox.moc，则需要修改 glbox.cpp 文件中的下面一行代码：

```
#include "glbox.moc"
```

因此，最好将其命名为 glbox.moc。之后，应该为每个.cpp 文件创建目标文件：

```
# g++ -c glbox.cpp -o glbox.o
# g++ -c glwindow.cpp -o glwindow.o
# g++ -c main.cpp -o main.o
```



实际上，当创建目标文件时，不是必须将你想使用的输出文件名称告诉 g++（或 moc）。如果只输入下面命令：

```
# g++ -c main.cpp
```

g++将认为你想要的目标文件名为 main.o。

当目标文件准备好后，该是将它们链接到一起形成可执行程序的时候了。这用下面命令来实现：

```
# g++ glbox.o glwindow.o main.o -o Listing-19.1 -lqt -lqgl -lGL
```

这个命令所产生的执行文件为 Listing-19.1。注意，需要链接 Qt 标准库、Qt 的 OpenGL 库和 MESA 提供的 OpenGL 库。

现在启动这个程序。如果所有都正确的话，你所看到的运行结果应该如图 19-1 所示。

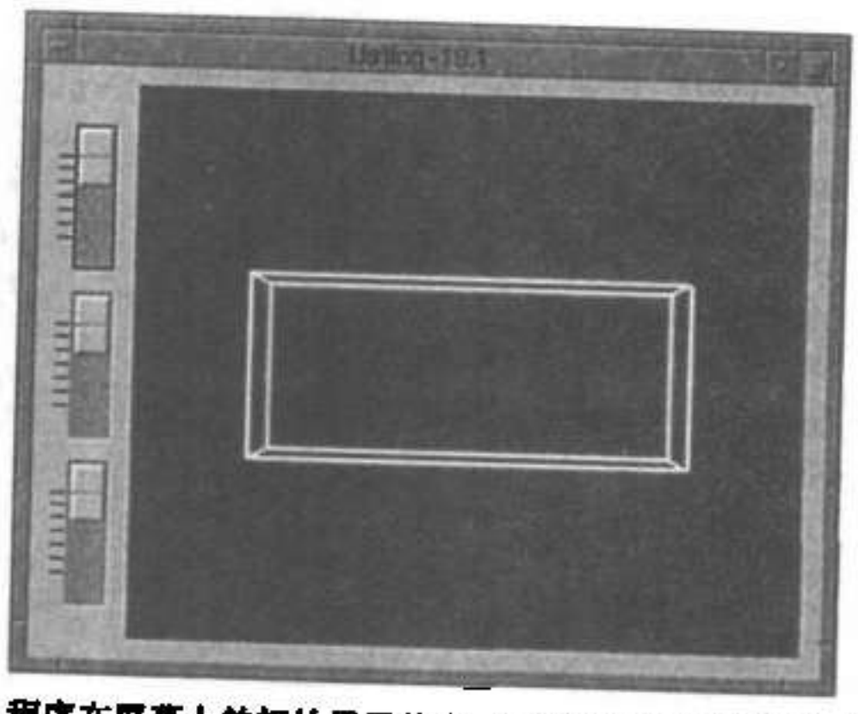


图 19-1 OpenGL 程序在屏幕上的初始显示状态，可以使用左边的滑块来改变盒子的视角

现在，试着拖动滑动框，来看视角的改变。你将看到类似于图 19-2 所示的图形。

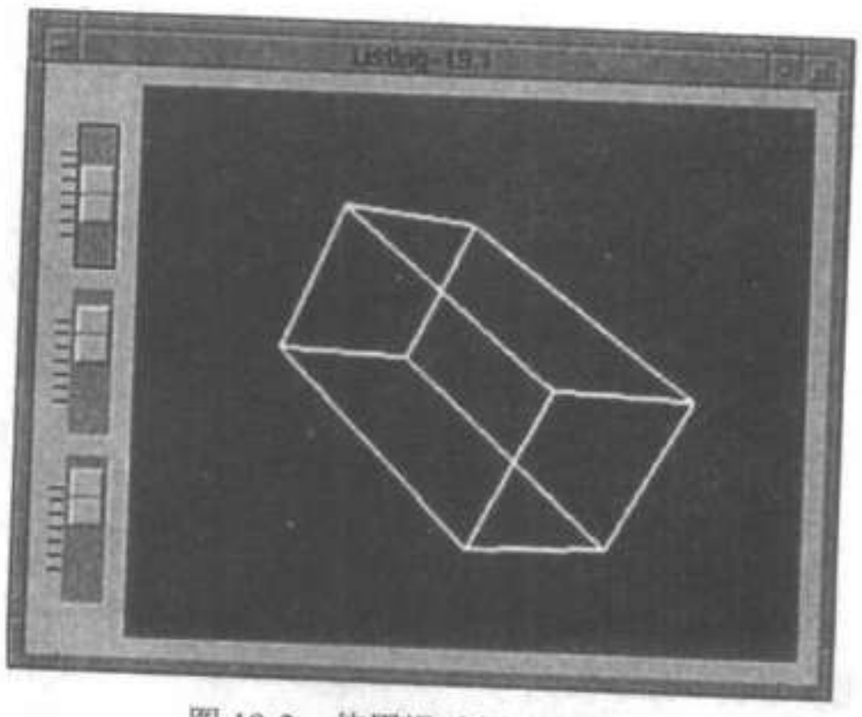


图 19-2 使用滑动框改变视角

尽管这个程序不是很有用，但它说明了将两个库，Qt 和 OpenGL，连接到一起是非常容易的。

19.4 小　结

OpenGL 最明显的缺点是它缺乏用户交互功能（你应该了解，OpenGL 从没有打算成为一个 GUI 程序）。如果 dramatic，你可能会说 Qt 的最大缺点是缺乏高性能的绘图操作。但是，将 Qt 与 OpenGL 库联合起来使用，就能够完全解决这一问题。将 OpenGL 用于图形操作，并将 Qt 用于与用户交互，其效果是非常好。

尽管 Open GL 有很强的图形功能，但是，在使用它之前还是应该再三考虑。毕竟 OpenGL 是一个全新的库，这意味着所有用户必须安装 MESA（或其他 OpenGL 实现）才能使用你的程序。如果没有 OpenGL 库，他们很可能不安装你的软件，因为这需要他们去获取和安装 OpenGL 库。因此，对于简单的图形功能，最安全的方法是坚持使用 Qpaint。

当然，只有当你计划分发软件源代码时才会出现这个问题。如果你计划只提供二进制代码，它只需要建一个静态链接库。采用这种方法，用户完全不必为下载和安装 OpenGL 库而烦恼。注意，缺省时，MESA 在编译时不创建静态库，必须向 configure 添加—enable – static 选项，之后再重新编译才能得到静态库。

19.5 问题与答案

问：事实上，我对这一学时中的内容理解不多。是因为哪些内容我没掌握吗？

答：这一学时假设你至少具备 OpenGL 程序设计方面的基本知识。没有这些知识，你将很难掌握所讨论的内容。从www.opengl.org可以得到很好免费的 OpenGL 文档。

问：当编译程序清单 19-1 时，编译器报告未找到任何 GL 库。但是，我肯定是已经正确安装了 MESA 库。

答：缺省时，MESA 将其库安装在/usr/local/lib 下。很可能是你未设置环境来搜索这个目录。但是，使用-L 选项能够很容易地改变这一点。例如，-L/usr/local/lib 将使编译器在/usr/local/lib 目录下查找库。

19.6 作　业

完成下面的问题和练习将有助于你牢记这一学时中所学的在 Qt 程序中怎样使用 OpenGL 方面的知识。

19.6.1 测验

1. 什么是 OpebGL?
2. 什么是 MESA?

3. 什么是 libqgl?
4. 什么时候应该使用 OpenGL?
5. OpenGL 的弱点是什么?
6. QGLWidget 的用途是什么?

19.6.2 练习

1. 在www.opengl.org站点中查找一本好的 OpenGL 教程，阅读并研究该教程。
2. 构造并测试$QTDIR/extensions/opengl/examples 目录中的例子。研究这些源代码，它向你介绍一些在 OpenGL 程序中怎样使用 Qt（反之亦然）的方法。
3. 向程序清单 19-1 添加一个功能，使用户能够改变视距。增加一个条，使它能够控制盒子到用户间的距离。

第 20 学时　创建 Netscape 插件

在这一学时，将学习怎样使用 Qt 创建浏览器插件，插件是为多种类型应用程序设计的小软件（UNIX 下为一个共享对象，Windows 下为一个 DLL 文件），它能够扩展这些程序的当前功能。目前流行的 Macromedia 公司的 Flash 插件就是一个例子，它使 Internet 浏览器能够处理使用 Flash 技术的 Web 站点（其更多信息请参看www.flash.com）。如果你已经安装了这个插件，当浏览器遇到插件能够处理的文件或数据流时，它将调用这个插件。要记住，一个程序的插件与其他程序的插件不兼容。

当为 Netscape Navigator 创建插件时，通常会使用 Plugin SDK（Software Development Kit，软件开发工具包），这可以从 Netscape 的 Web 站点下载，并使用 Plugin SDK 所提供的 API（Application Programming Interface，应用程序编程接口）编写插件。但是，Plugin SDK 所提供的 API 相当难用，这就是 Qt 产生的原因。Qt 包含几个类，用于创建 Netscape 插件。通过使用这些类，你不必使用标准的 Plugin API，因此，使编写 Netscape 插件变得更加容易。

这一学时重点讨论怎样使用 Qt 插件类创建 Netscape 插件，但不介绍怎样使用 Netscape Plugin SDK。

20.1　建立插件开发环境

在使用 Qt 开发 Netscape 插件之前，需要先完成几件事情。这一节介绍这些实现步骤。

20.1.1　获得 Netscape Plugin SDK

尽管不直接使用 Netscape 提供的 Plugin SDK，但是 Qt 中的插件扩展用到这一 SDK 中的部分内容。因此，你需要获得该软件。

这可以从 Netscape 的 Web 站点下载（http://home.netscape.com/comprod/development_partners/plug_api/）。要确保得到适合你系统的正确版本（Windows 或 UNIX）。当下载完后，需要解开文件并将几个文件拷贝到$QTDIR/extensions/nsplugin/src 目录下。首先，解开文件：

```
# tar xvfz unix-sdk-3.0b5.tar.Z
```

这将解开整个文件，并将所有文件放置在 PluginSDK30b5 目录下。



像通常一样，其版本号很可能已经改变。在编写本书时，最新版本为 3.0b5。如果发生变化，这将影响文件名和目录名。

现在，你需要将这些文件中的几个文件拷贝到$QTDIR/extensions/nsplugin/src 目录下。这可按下面方法来实现：

```
# cd PluginSDK30b5/common
# cp npunix.cpp $QTDIR/extensions/nsplugin/src
# cp npwin.cpp $QTDIR/extensions/nsplugin/src
# cd ../include
# cp * $QTDIR/extensions/nsplugin/src
```

在此之后，可以删除 PluginSDK30b5 目录——之后不再需要它。

20.1.2 编译 Qt 的 Netscape 插件扩展

缺省时，不编译 Qt 的 Netscape 插件扩展——你需要手工进行编译。但是，如果没有上一节中所拷贝的文件，则不能编译它。编译并不困难，使用 cd 命令将当前目录切换到 $QTDIR/extensions/nsplugin/src，并从该目录下运行 make。例如：

```
# make
```

你将看到以下输出内容：

```
g++ -c -I/usr/local/qt/include -I/usr/X11R6/include -pipe -02 -fPIC
-o qnp.o qnp.cpp
/usr/local/qt/bin/moc ../../../include/qnp.h -o moc_qnp.cpp
g++ -c -I/usr/local/qt/include -I/usr/X11R6/include -pipe -02 -fPIC
-o moc_qnp.o moc_qnp.cpp
rm -f ../../../lib/libqnp.a ; ar cqs ../../../lib/libqnp.a qnp.o moc_qnp.o
```

这将产生一个静态库，叫做 libqnp.a。将这个文件拷贝到$QTDIR/lib 下，这样你在编译插件软件时不必每次都指定这个库所在目录。

20.2 Qt 的 Netscape 插件类

首先，你需要理解 Qt 插件类的工作机制。这一节介绍所有这些类，讨论怎样使用和什么时候使用它们。注意，这里只介绍每个类中最重要的成员函数。欲了解这里未提到的其他函数请参考 Qt Reference Document。

20.2.1 QNPlugin：插件核心

QNPlugin 是插件的核心，这个插件中没有其他更重要的对象了。对于用 Qt 所创建的每一个插件，都需要创建一个 QNPlugin 子类，并重新实现几个浏览器能够调用以获得关于这个插件各种信息的函数。

一、理解 getMIMEDescription()函数

当需要了解 MIME 类型和插件能够处理的文件扩展名时，浏览器需要调用 QNPlugin::getMIMEDescription()函数。

浏览器使用 MIME 类型判断它能够处理哪种类型的数据。由 Web 服务器告诉浏览器一

个文件或数据流的 MIME 类型。之后，浏览器检查其配置了解是否为这种 MIME 类型定义有处理程序（一个程序或者一种插件）。如果有处理程序，则使用它处理数据。

一个 MIME 类型由主类型和次类型组成，它们由斜杠(/)分隔。例如，MIME 类型 text/plain 表示纯文本类型，MIME 类型 image/gif 表示 GIF 图像类型。

然而，为了定义一种新的 MIME 类型，你需要有权访问 Web 服务器的配置文件，但这不是你通常所具有的。幸运的是，查看文件扩展名也能够判断文件类型。例如，文本文件通常以.txt 结尾，GIF 图像文件以.gif 结尾。

这里是一个 QNPlugin::getMIMEDescription()函数实现的例子：

```
const char *getMIMEDescription( )
{
    return "image/gif:gif:GIF Image";
}
```

如你所看到的，这个字符串被分为 3 部分，由冒号(:)分隔。第 1 部分表示 MIME 类型，第 2 部分表示文件扩展名，最后一部分表示类型描述。注意，文件扩展名可以包含多个。如果你想为一种类型注册多个文件扩展名，则需要用逗号(,)将它们进行分隔。

二、理解 getPluginNameString()函数

QNPlugin::getPluginNameString ()函数以字符串形式返回插件名称。这个字符串出现在 Netscape Navigator 的 About Plug-Ins 页面中的 Description 列。从 Help 菜单（位于浏览器窗口的右端）中能够访问 About Plug-Ins 页面。也可以在 URL 域中输入 about:plugins 来访问它。下面是一个 QNPlugin::getPluginNameString ()函数实现的例子：

```
const char *getPluginNameString ( ) const
{
    return "This is a full name of my plugin";
}
```

三、理解 getPluginDescriptionString()函数

QNPlugin::getPluginDescriptionString()函数也返回显示在 About Plug-Ins 页面上的字符串。但是，这个函数返回插件的简短描述，而不是插件名称。下面是一个 QNPlugin::getPluginDescriptionString ()函数实现的例子：

```
const char *getPluginDescriptionString ( ) const
{
    return "Here is a short description of my plugin";
}
```

四、理解 newInstance()函数

浏览器使用 QNPlugin::newInstance ()函数读取 QNPInstance 类的新对象（也就是你已经创建的 QNPInstance 类的子类），如果你的 QNPInstance 子类叫做 MyPluginInstance，QNPlugin::newInstance()函数将像下面这样：

```
QNPInstance *newInstance ( )
```

```
{
    return new MyPluginInstance;
}
```

20.2.2 QNPInstance：浏览器和插件之间的链接

如前所述，没有比 QNPlugin 更重要的对象。然而，QNPInstance 对象是通过调用 QNPlugin::netInstance()函数创建的。在 QNPlugin 对象创建和 QNPlugin::netInstance()函数调用之后，QNPInstance 负责插件和浏览器之间的通信，我们现在来学习这一内容。

一、理解 newWindow()函数

QNPInstance::newWindow()返回一个指向 QNPWidget 子类对象的指针。当插件显示在屏幕上时，这个函数被浏览器调用一次。下面是该函数的一个实现例子：

```
QNPWidget *newWindow( )
{
    return new MyPluginWidget;
}
```

在这里，QNPWidget 的子类叫做 MyPluginWidget。

二、理解 newStreamCreated()函数

浏览器使用 QNPInstance::newStreamCreated ()将数据可用信息告诉插件。使用这个函数，也可以告诉浏览器你所需的数据是作为文件形式还是流方式。如果数据被接受之后，QNPInstance::newStreamCreated ()函数返回 TRUE。这里是一个实现例子：

```
bool newStreamCreated ( QNPStream*, StreamMode &smode)
{
    smode = AsFileOnly;
    return TRUE;
}
```

在这个例子中，插件接受数据，并告诉浏览器它需要以文件形式提供数据。在这里，由于浏览器已经被告知以文件形式提供数据，所以，后面将调用 QNPInstance::streamAsFile()。这个函数所包含的代码处理文件中的数据。

如果将 smode 设置为其他 3 种选项（Normal、Seek 或 AsFile）之一，数据可能将以流形式接收。之后，浏览器应该调用 QNPInstance::writeReady() 和 QNPInstance::write()，前者用来查询实例能够从指定流中接收的最小数据量，而后者则负责实际处理（或调用适当的处理函数）。

根据你所处理的数据不同，QNPInstance::streamAsFile()、QNPInstance::writeReady() 和 QNPInstance::write()中的代码也会有很大的变化。因此，这一节不再介绍这些函数的实现实例。然而，Qt Reference Document 中包含一些关于这些函数的有趣信息。

20.2.3 QNPWidget：创建插件可视区域

QNPWidget 子类中应该包含插件的事件和绘图代码。QNPWidget 是 QWidget 的一个子

类，因此可以把它看做与其他 Qt 部件相同。实际上，这个类中的代码常常只是常规的 Qt 程序设计代码（因此你并不需要任何例子）。

20.2.4　QNPStream：从浏览器接收数据流

只有当你使用流而不是文件方式接收数据时才需要实现 QNPStream 子类。QNPStream 是一个数据流的抽象，QNPInstance::writeReady() 和 QNPInstance::write()将它用于流数据。但是，这个类的实现也随所处理的数据不同而有很大的变化。Reference Document 中包含有关于这个类及其成员的有用信息。

20.3　创建第一个 Netscape 插件

这一节提供一个非常简单的基于 Qt 的插件例子。你将学习怎样编译这个例子，怎样使 Netscape 知道这个插件，最后，学习怎样去测试它。

20.3.1　研究代码

现在准备编写一个插件。程序清单 20-1 是一个简单的例子：

程序清单 20-1　　一个非常简单的基于 Qt 的 Netscape 插件

```
1: //We need to include qnp.h to get the
2: //plugin definitions:
3: #include <qnp.h>
4: #include <qpainter.h>
5:
6: //This is our QNPWidget sub-class. It will
7: //create the actual plugin widget:
8: class MyPluginWidget : public QNPWidget
9: {
10: public:
11:
12: //We reimplement the paintEvent() function
13: //to take care of the painting:
14: void paintEvent(QPaintEvent* event)
15: {
16:     QPainter p( this );
17:     p.setClipRect( event->rect() );
18:     int w = width();
19:     p.drawRect( rect() );
20:     p.drawText( w/8, 0, w-w/4, height(), AlignCenter,
➥ "Your first Qt-based plugin!" );
21: }
22:
23: };
24:
25: //This is our QNPInstance sub-class. This simple
26: //version only includes the newWindow() function,
27: //which returns an object of the MyPluginWidget
28: //class:
29: class MyPluginInstance : public QNPInstance
```

```
30: {
31: public:
32:
33: QNPWidget *newWindow()
34: {
35:     return new MyPluginWidget;
36: }
37:
38: };
39:
40: //MyPlugin is the heart of our plugin.
41: //Its members are called by the browser
42: //to get various information about the plugin
43: //and to get an object of the MyPluginInstance
44: //class:
45: class MyPlugin : public QNPlugin
46: {
47: public:
48:
49: QNPInstance *newInstance()
50: {
51:     return new MyPluginInstance;
52: }
53:
54: const char *getMIMEDescription() const
55: {
56:     return "basic/very:bas:A very basic plugin";
57: }
58:
59: const char *getPluginNameString() const
60: {
61:     return "A very basic Qt-based Plugin";
62: }
63:
64: const char *getPluginDescriptionString() const
65: {
66:     return "A very simple Qt-based plugin that can't do anything useful";
67: }
68:
69: };
70:
71: //The extern function that will feed the
72: //browser with an object of the MyPlugin
73: //class:
74: QNPlugin* QNPlugin::create()
75: {
76:     return new MyPlugin;
77: }
```

如你所看到的，这是一个非常简单的例子。因为函数非常短，所以将函数定义放置在类定义中（第8行到第23行和第29行到第69行，这也叫做内嵌式程序设计）。这使得代码更加容易理解。这个程序中的大部分在“Qt的Netscape插件类”一节中已经讨论过。paintEvent()函数（第14行到第21行）也没有提供任何新内容，因此不再多讨论它。但是，注意所用到的MIME类型basic/very及文件扩展名.bas（对于basic）（第56行）。

实际上，不需要进一步解释程序清单 20-1 这一事实也就是你为什么应该使用 Qt 开发

Netscape 插件的一个重要原因。相反，如果你选择使用 Netscape Plugin SDK 所提供的标准 API，就需要学习和理解那个 API 中的所有函数。现在，你需要做的唯一件事是学习怎样实现插件和浏览器之间的通信。对于实际的插件实现，能够使用通常的 Qt 类，这时你应该很熟悉它们。

20.3.2　编译和安装插件

如前所述，一个插件是一个共享对象（UNIX）或一个 DLL 文件（Windows）。为了使它成为共享对象，你应该用一种与编译其他 Qt 程序不同的方法来编译它。

首先，需要从源代码中产生目标文件：

```
# g++ -c 20lst01.cpp -o 20lst01.o
```

这建立一个名为 20lst01.o 的目标文件。之后，需要与其他库一起链接这个目标文件，并从中产生一个共享对象。为此，执行下面命令：

```
# g++ -L/usr/local/qt/lib -L/usr/X11R6/lib 20lst01.o -lqt -lXext -lX11
-lm -L/usr/local/qt/lib -lqnp -lXt -lm -shared -o 20lst01.so
```

注意-lqnp 选项，它告诉链接器（可能为 ld）与在前面“建立插件开发环境”一节中所创建的 libqnp 库一起链接。也要注意-shared 选项。这一选项使命令的输出为一个共享对象。

这将产生一个叫做 20lst01.so 的文件。你应该将这个文件拷贝到 Netscape 安装下的插件目录中，它可能为/usr/lib/netscape/plugins、/usr/local/netscape/plugin、/usr/netscape/plugin 或/opt/netscape/plugins 之类的目录。为了拷贝文件，执行下面命令：

```
# cp 20lst01.so /usr/lib/netscape/plugins
```



如果想了解更多关于编译和链接方面的信息，可以阅读该系列丛书中的另外一本，Sams Teach Yourself Linux Programming in 24 Hours，作者为 Warren W. Gay(ISBN 0-672-31582-3)。它详细讨论关于使用 g++(gcc)和 ld 进行编译和链接。

如果 Netscape 已经运行，你需要重新启动它。之后，在浏览器窗口里从 help 菜单中选择 About Plug-Ins 项。现在，如果所有内容都正确的话，你应该看到上一页中的插件信息。如图 20-1 所示。

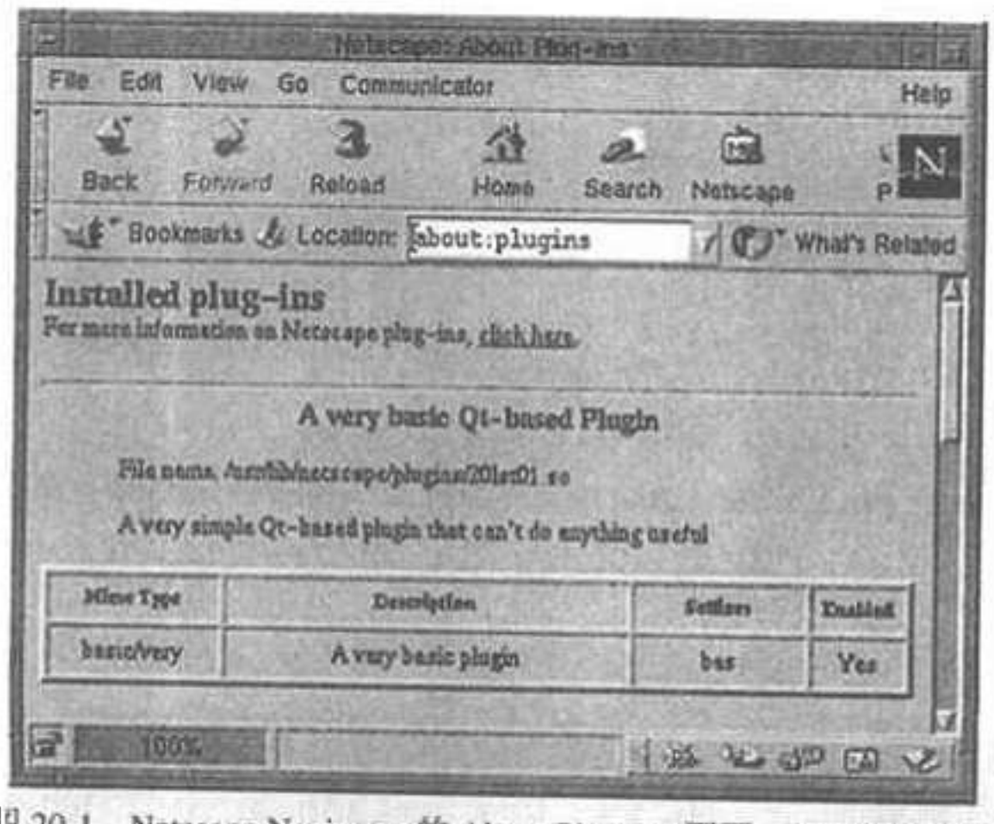


图 20-1　Netscape Navigator 的 About Plug-Ins 页面，显示插件例子

“A very basic Qt-based Plugin”标题来自于 QNPlugin::getPluginNameString()函数。文件名下的描述来自于 QNPlugin::getPluginDescriptionString()函数，表中的 3 个域在 QNPlugin::getMIMEDescription()函数中定义。

20.3.3 测试插件

因为这个简单的插件不处理任何数据，所以不要使用具有正确扩展名（.bas）的文件来测试插件。相反，将使用 HTML 标记<EMBED>来调用插件。因为这不是一本 HTML 书籍，所以这一节只讨论<EMBED>标记，这足以教会你怎样使用它来显示插件。

为了测试插件，首先需要创建一个 HTML 文档。这里是一个例子：

```
<HTML>
<HEAD>
<TITLE>A Very Basic Qt-Based Plugin Called With the EMBED Tag</TITLE>
</HEAD>
<BODY BGCOLOR="#ffffff">
<CENTER>
<EMBED TYPE=basic/very WIDTH=300 HEIGHT=100>
</CENTER>
</BODY>
</HTML>
```

现在，如果在 Netscape 中打开这个文档，插件将绘制一个 300×100 像素的矩形（由 EMBED 标记定义），矩形内将显示“Your first Qt-based plugin!”文本。如图 20-2 所示。

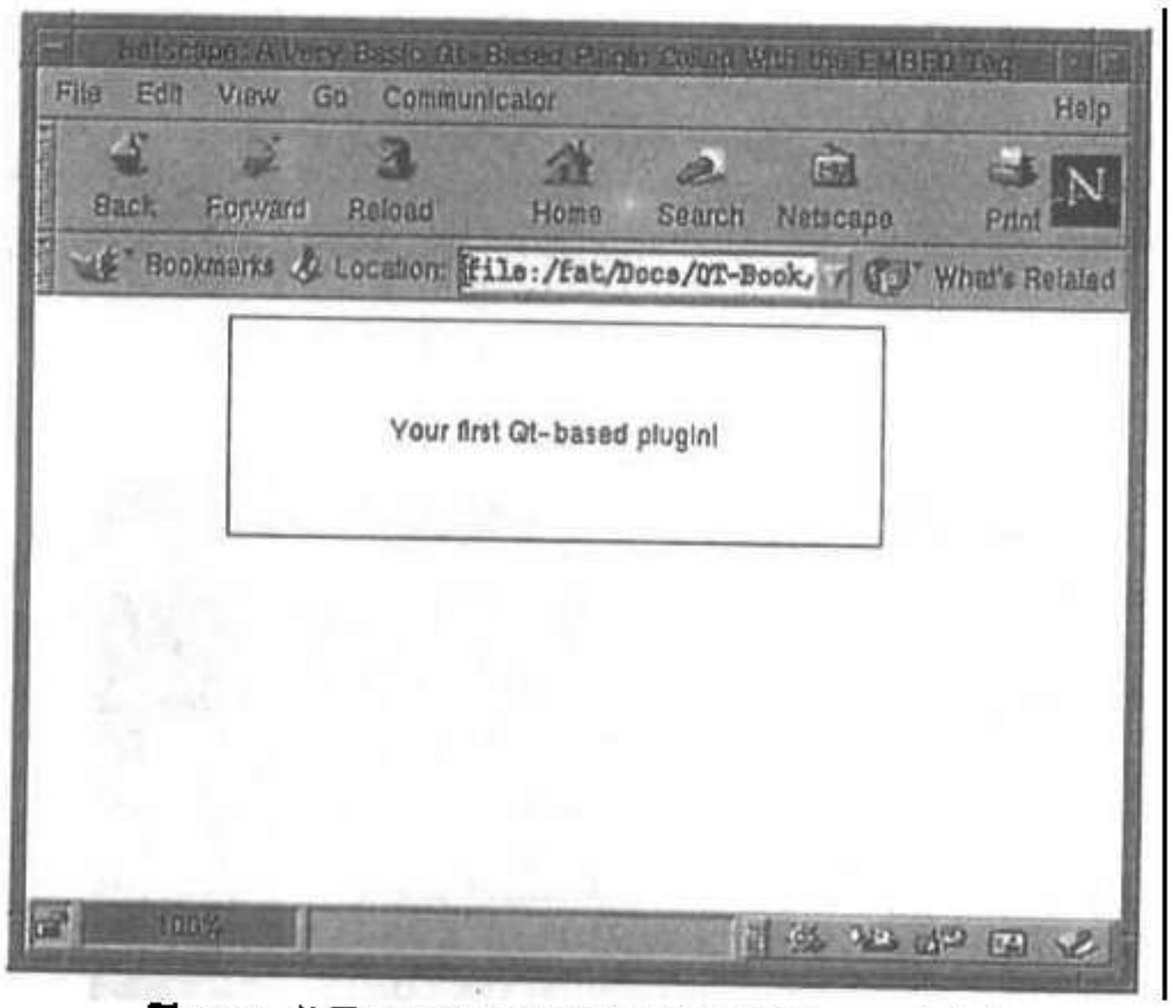


图 20-2 使用 EMBED 标记将插件嵌入到 HTML 文档中

这个例子没什么用处，它只是绘制一个包含一些文本的矩形。但是，假设你已经创建了一个显示 AVI 电影的插件，就能够很容易地像下面这样将 AVI 电影嵌入到 HTML 文档中：

```
<EMBED SRC="movie.avi" WIDTH=320 HEIGHT=200 AUTOSTART=true LOOP=TRUE>
```

需要了解<EMBED>标记的更详细信息时，你可以查阅 Internet 上很多免费的 HTML 教程。

20.4　小　结

使用 Qt 开发 Netscape 插件能够使程序设计的很多方面变得更加容易。选择 Qt 而不是 Netscape Plugin SDK 所提供的标准 API 能够节省大量的时间。Netscape Plugin SDK 所提供的 API 相当复杂，而且难以使用。现在，你应该清楚地认识到使用 Qt 时情况就不同了。

在 Qt 安装的 extensions/nsplugin/examples 目录下有 3 个有趣的插件实例。或许这 3 个中最有用的是 qtimage 插件。它扩展 Netscape 的功能，使它能够显示 Qt 所支持的图像格式！希望你看一下这个插件的源代码。它说明使用 Qt 创建有用的插件是多么简单。

当创建 Qt 插件时，你将发现通过 Netscape 进行调试相当困难。因此，应该首先将插件编译为常规的 Qt 程序，之后，当它通过测试并能够使用时，再将它转换为插件。

20.5　问题与答案

问：当我链接 libqnp 库时，编译器报告未找到任何具有这个名称的库。这是什么原因？

答：你可能忘了将这个库从$QTDIR/extensions/nsplugin/src 目录拷贝到$QTDIR/lib。如果不这样做将找不到这个库。

问：当执行命令产生共享对象时，编译器报告不能找到指定要链接的库。这是什么原因？

答：可能是库不在编译器所搜索的目录下。试着用手工方式找到这个库（用 find 或 locate）。如果能够找到，用-L 选项告诉编译器去搜索那个目录。如果未找到库，将它从链接列表中删除，看插件能否正常工作。如果不能工作，可从分发程序或者 www.freshmeat.net 站点中查找这个库。

问：当编译 qtimage 插件时，编译器报告不能找到 qimgio 库，到哪里去查找这个库？

答：这个库能从 Qt 安装目录下的 extensions 目录中找到。缺省时，它未被编译和安装——你必须手工处理。

20.6　作　业

完成下面的问题和练习将有助于你牢记这一学时中所学的 Qt 插件类以及怎样使用它们。

20.6.1　测验

1. 为什么应该使用 Qt 去开发 Netscape 插件？
2. QNPlugin::getMIMEDescription()函数的功能是什么？

3. 什么是 MIME 类型？
4. 假设已经创建了一个显示 BMP 图像的插件。你打开一个具有很多 BMP 图像的文档。有多少个 QNPlugin 对象被创建？
5. 在第 4 个问题中，创建有多少个 QNPInstance 对象？
6. QNPInstance::newWindow()函数的用途是什么？
7. 什么时候浏览器调用 QNPInstance::newStreamCreated()函数？
8. 在 QNPWidget 子类中能够找到哪一类代码？
9. 什么时候需要创建 QNPStream 子类？
10. HTML 标记<EMBED>的用途是什么？

20.6.2　练习

1. 编译、安装和测试 qtimage 插件。
2. 编写一个插件，它从文件中读取文本（使用特殊的文件扩展名）并将文本显示（或绘制）在浏览器窗口上。一个想法是显示文本的方式应该超过通常的 HTML，插件将因此而非常有用（注意，这样做需要学习 Reference Document）。例如，用插件将文本显示为圆状之类。

第五部分

改善程序性能

第 21 学时　Qt 程序国际化

第 22 学时　可移植性

第 23 学时　调试技术

第 24 学时　使用 Qt 构造程序

第 21 学时　Qt 程序国际化

为了使 Qt 的程序能够被更多人更加容易地使用，你需要做的事情之一就是使它们能够使用其他语言。尽管许多人将英语作为第二语言，但是，通常发现如果使用母语人们能够更容易地全面理解程序。如果英语不是你的母语，你将发现，为了查找程序中一两个不懂的单词是多么地令人感到沮丧。

所幸的是，Qt 带有很多工具和函数，它们使程序国际化工作变得更加简单。这不仅包括语言翻译，也包括字符编码、输入技术和显示转换。

在这一课，将学习能够帮助你创建国际化 Qt 应用程序的函数和工具。

21.1　QString 的重要性

首先，你应该使用 QString 处理所有用户可见文本（也就是以任何方式显示给用户的文本）。

与 char 相比，QString 内部使用 Unicode 编码。这使 QString 能够毫无问题地处理世界上的所有语言。相反，如果使用 char，这很可能造成一些系统无法以其希望的方式显示文本。通过使用 QString 显示用户可见文本，能够确保不会出现这种问题。

使用 QString 的另一个重要原因是速度。所有向用户显示文本的 Qt 函数均以 QString 做参数。尽管这些函数也可以使用 char，但这样做将强制进行从 char 到 QString 的类型转换，这一转换要占用一定的 CPU 时间。使用 QString 代替 char，就可以避免这一转换。但是，仅仅将几个 char 实例修改为 QString，你将不会觉察到速度方面的变化。只有当程序被强制进行大量的 char 到 QString 转换时，速度变化才会明显。

对于用户完全看不到的字符串（也称做程序员空间文本），可以安全使用 char。

21.2　创建翻译文件

在 Qt 中创建翻译文件时，需要执行多个操作。其中包括修改源代码和使用 Qt 实用程序创建特殊的翻译文件，它用于查找翻译特殊的字或词语。在这一节，将学习创建翻译文件所需的所有操作。

21.2.1　使用 tr()函数

QObject::tr ()函数必需用在程序中所有需要翻译字符串的地方。这个函数非常容易使

用，只需将需要翻译的字符串作为参数传递给它。因为它是 QObject 的一个成员函数，所以它包含在所有继承 QObject 的类中。例如，如果你想翻译标签 Password，在一个口令登录部件中，可以使用类似下面的代码：

```
QLabel *label = new QLabel( tr("Password:"), this );
```

这使 tr()函数去搜索 Password:字符串的翻译。如果查找到翻译，则使用那个字符串。如果未查找到其翻译，则使用 Password:。但是，当然需要翻译文件才能使它工作（参看下面一节）。如果没有翻译文件，tr()函数调用将总是返回 Password:。

如果要翻译的文本不是 QObject 的子类（例如，它在一个全局函数内），你可以使用其他合适类中的 tr()函数。例如：

```
void MyGlobalFunction( )
{
    QLabel *label = new QLabel( MyObjectSubClass::tr(
      "Text to be translated:"), parent);
}
```

尽管上面用到的 tr()函数不是 QObject 子类的成员，但是它和前一个例子具有相同的功能。注意函数名和类名是虚构的。

因此，通过对程序中所要显示给用户的所有文本使用 tr()函数，就能够保证在需要的时候执行翻译操作。但是，如前所述，你需要创建翻译文件，它存储 tr()函数可以使用的翻译信息。

21.2.2　使用 findtr 实用程序提取翻译文本

为了使用 tr()函数，你需要创建翻译文件。这些文件像程序翻译所使用的数据库一样工作——每种语言一个文件。这一节介绍在这一过程中所需要执行的所有操作步骤。



不是必需使用这一节中所用的文件名转换操作。但是，建议你这样做，以遵守 Qt 翻译文件标准。这使其他人能够更容易地理解你程序的组织方式。

首先，你需要查找程序中所有的 tr()函数调用，并将它们提取到一个 PO 文件（具有.po 扩展名的文件）。这由 findtr 实用程序实现，它位于$QTDIR/bin 目录下。这个程序在一个或多个文件中查找 tr()函数调用，并在 PO 文件中为每个调用添加一项。

之后，你需要编辑文件中的每一项，使它们都有一个合法的翻译。来看程序清单 21-1 所示的口令部件这个例子：

程序清单 21-1　　一个典型的口令部件

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qlabel.h>
4: #include <qlineedit.h>
5:
6: class MyWidget : public QWidget
7: {
```

```
 8: public:
 9:         MyWidget();
10: private:
11:         QLabel *label;
12:         QLineEdit *edit;
13: };
14:
15: MyWidget::MyWidget()
16: {
17:     resize( 200, 50 );
18:
19:     label = new QLabel( "Password:", this );
20:     label->setGeometry( 10, 10, 60, 30 );
21:
22:     edit = new QLineEdit( this );
23:     edit->setGeometry( 70, 10, 120, 30 );
24:     edit->setEchoMode( QLineEdit::Password );
25: }
26:
27: main( int argc, char **argv )
28: {
29:     QApplication a( argc, argv );
30:     MyWidget w;
31:     a.setMainWidget( &w );
32:     w.show();
33:     return a.exec();
34: }
```

这个程序创建一个左边具有一个标签（第 19、20 行），右边具有一个 QLineEdit 对象（第 22、23 和 24 行）的小窗口——一个典型的用于读取口令的部件（如图 21-1 所示）。

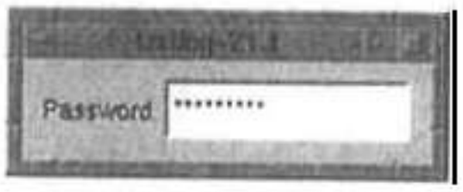


图 21-1　一个典型的读取口令部件

对于所有不精通英语的用户来说，如果将 Password:标签修改为他们所使用的语言，不是能起到更好的提示作用吗？所幸的是，这非常容易。

首先，需要实现前面讨论的 tr()函数。只需将下面这行：

```
label = new QLabel( "Password:", this );
```

修改为：

```
label = new QLabel( tr("Password:"), this );
```

现在，需要创建拥有 Password:标签翻译的 PO 文件。执行下面命令：

```
# findtr 21lst01.cpp > 21lst01.po
```

这个命令将在 21lst01.cpp 中查找 tr()函数调用，并将每一个调用输出一项到 21lst01.po 中。如果你按照前面的指导在程序清单 21-1 中实现 tr()函数，findtr 输出将像下面这样：

```
# This is a Qt message file in .po format. Each msgid starts with
# a scope. This scope should *NOT* be translated - eg. "Foot::Bar"
# would be translated to "Pub", not "Foo::Pub".
msgid ""
```

```
msgstr ""
"Project -Id-Version: example-Qt-message-extraction\n"
"POT-Creation-Date: 1999-02-03 15:38+0200\n"
"PO-Revision-Date: 1999-02-03 15-38+0200\n"
"Last-Translator: \n"
"Content-Type: text/Plain; charset=iso-8859-1\n"

#: 21lst01.cpp:19
msgid "MyWidget::Password:"
msgstr ""
```

前 10 行包含的内容为所有情况下 findtr 所输出的标准数据。但是，其后的数据则依赖于 findtr 所查找的文件。在这里，最后 3 行很重要。来看下面一行：

```
#: 21lst01.cpp:19
```

这一行告诉你，在 21lst01.cpp 文件中的第 19 行发现 tr()函数调用。之后，看下面一行：

```
msgid "MyWidget::Password:"
```

这是翻译项标识，MyWidget 为翻译项范围（也就是调用 tr()函数的类名）。Password 为翻译项键值（也就是应该被翻译的文本）。之后，有一个翻译字符串行：

```
msgstr ""
```

在这一行，你应该插入代替键值的字符串（在这里为 Password:）。但是，通常不直接编辑原始的 PO 文件，常用方法是拷贝原始 PO 文件，每种语言拷贝一份，之后编辑拷贝文件。例如，如果想做这个部件的瑞典语翻译，你应该将原始 PO 文件拷贝到另一个合适的文件名，如 21lst01_se.po，之后对该文件做修改。这里，21lst01_se.po 中的最后 3 行应该修改为：

```
#: 21lst01.cpp:19
msgid "MyWidget::Password:"
msgstr "Lösenord:"
```

现在，如果在程序中使用这个翻译，对 tr("Password:")的每个调用都将返回字符串 Lösenord:。



很容易使 findtr 搜索多个文件中的 tr()函数调用，而不需要为它们中的每个文件分别执行一次命令。例如，下面命令将搜索以.cpp 或.h 结尾的所有文件，并将结果输出到一个叫做 MyClass.po 的文件中：

```
# findtr *.cpp *.h >MyClass.po
```

21.2.3　使用 msg2qm 实用程序创建二进制翻译文件

现在，已经创建了一个有效的 PO 文件（21lst01_se.po），你需要从它产生一个二进制数据库文件。这由 Qt 实用程序 msg2qm 实现。使用二进制数据库代替常规文本文件，能够使翻译更快。更重要的是，必须这样做才能使翻译无论在什么地方都能工作。

与 findtr 一样，msg2qm 也位于$QTDIR/bin 目录下。它带两个命令行参数：要使用的 PO 文件和输出结果的存储文件名（也就是二进制数据库文件）。因此，为了给上一节中所建立

的 21lst01_se.po 创建一个二进制数据库文件，应该执行下面命令：

```
# msg2qm 21lst01_se.po 21lst01_se.qm
```

这创建一个 21lst01_se.qm 文件，之后，在程序中可以使用它来检索翻译信息。

21.2.4　用 mergetr 合并修改

在开发程序时，文本很可能需要修改。使用 findtr 能够很容易地修改原始 PO 文件。但是，也需要编辑与所有其他语言相关的 PO 文件中的翻译项。如果已经翻译为多种语言，手工完成这些工作是非常耗时的。

幸好，mergetr 实用程序能够帮助解决这一问题。它只在原始（和修改后的）PO 文件中搜索修改，之后，相应地编辑其他 PO 文件。例如，随着程序变大，行号很可能会发生改变。因此，PO 文件中的行号就不正确。然而，mergetr 能够为你做修正。

为了使用 mergetr 修改 PO 文件，首先需要像下面这样修改原始 PO 文件：

```
# findtr *.cpp *.h > MyClass.po
```

MyClass.po 将包含修改后的信息。之后，可以使用 mergetr 修改与所有语言相关的 PO 文件。这里是一个例子：

```
# mergetr MyClass.po MyClass_fr.po
```

它修改 MyClass_fr.po 文件。如果任何行号发生变化，它们都将被修改。如果任何翻译被删除，它们将在 MyClass_se.po 文件中被注释掉（使用#号）。此外，如果添加新的翻译项，它们也将被添加到 MyClass_se.po。

记住，如果你修改一个翻译（例如，删除 Password:中的冒号），findtr 将把此解释为一个新项，它不会把它当作旧的修改项。mergetr 也是这样。如果在原始 PO 文件中查找到与旧项不完全匹配的翻译项，它将把这解释为新项（新项的翻译字符串总为空，因此，你需要重新再次插入它们）。因为 mergetr 不能查找到完全匹配的旧项，它将被注释掉（在相关语言的 PO 文件中）。

mergetr 自动修改行号，它删除（或注释掉）所有不再使用项，如果在原始 PO 文件中有任何新项，它们也将被自动添加到相关语言的 PO 文件中。然而，它将被修改的翻译项也解释为新项。

21.3　在程序中实现翻译功能

在二进制翻译文件创建和准备好后，即可在程序中使用它们，这不是自动完成。但是，QTranslator 类使这一过程变得非常简单。下面是具体操作步骤：

(1) 创建 QTranslator 对象。

(2) 告诉 QTranslator 对象使用哪个二进制文件（用 msg2qm 实用程序所创建的二进制文件）。

(3) 将新的翻译器告诉 QApplication。

(4) 在类声明中调用 Q_OBJECT 宏，并对头文件使用 moc。

首先，创建 QTranslator 对象。之后，告诉它使用哪个二进制文件，并告诉 QApplication 对象一个新的翻译器可以使用。还需要像下面这样包含 qtranslator.h 文件，以访问 QTranslator

类声明：

```
#include < qtranslator.h >

QTranslator translator( this );
translator.load( "21lst01_se.qm", "." );
qApp->installTranslator( &translator );
```

第 1 行创建 QTranslator 对象（translator），并将其父对象设置为 this 指针（一个指向当前定义类中还未创建的对象指针）。在第 2 行，调用 translator.load()函数告诉 QTranslator 对象使用哪个二进制翻译文件。传递给该函数的第 1 个参数为二进制翻译文件名称，第 2 个参数定义程序应当在哪个目录中搜索这个文件。在这里，搜索当前目录(.)。在第 3 行，调用 QApplication:: installTranslator()函数将这个翻译器告诉 QApplication 对象。翻译器的内存地址作为参数传递。

因为翻译函数 tr()被 moc 所实现，所以，也需要调用 Q_OBJECT 宏创建一个类声明的 MOC 文件（并且要像通常一样，在 CPP 文件中包含 MOC 文件）。

在做这些修改之后，程序清单 21-1 中的类声明和定义看起来应当与程序清单 21-2 相同：

程序清单 21-2　　一个国际化的 Password 部件

```
1: class MyWidget : public QWidget
2: {
3:         Q_OBJECT
4: public:
5:         MyWidget();
6: private:
7:         QLabel *label;
8:         QLineEdit *edit;
9: };
10:
11: #include "21lst01.moc"
12:
13: MyWidget::MyWidget()
14: {
15:         resize( 200, 50 );
16:
17:         QTranslator translator( this );
18:         translator.load( "21lst01_se.qm", "." );
19:         qApp->installTranslator( &translator );
20:
21:         label = new QLabel( tr("Password:"), this );
22:         label->setGeometry( 10, 10, 60, 30 );
23:
24:         edit = new QLineEdit( this );
25:         edit->setGeometry( 70, 10, 120, 30 );
26:         edit->setEchoMode( QLineEdit::Password );
27: }
```

然而，实际上你应当从这个文件中产生两个文件。现在，我们来测试翻译功能。首先，对类声明使用 moc：

```
# moc 21lst01.h -o 21lst01.moc
```

要将 MOC 文件包含在 CPP 文件中，之后，重新编译程序清单 21-1 编辑后的版本：

```
# g++ -lqt 21lst01.cpp -o Listing-21.1
```

如果已经使用 msg2qm 实用程序创建了 QM 文件（从 PO 文件），你现在就能够测试程序。但是，因为向源代码中添加了几个新行，所以，PO 文件中的行号不再相符。但翻译仍将继续工作，并且，在这里只有一个翻译，它没有任何影响。

然而，在具有很多翻译的大型 PO 文件中，行号相符就更加方便，它能够更加容易地找到相关项。因此，无论你是使用基于 21lst01_se.po 文件旧版本的 QM 文件，还是从修改后的 21lst01_se.po 版本中重新创建它，运行程序清单 21-1 中的程序都将显示出图 21-2 所示的显示结果。

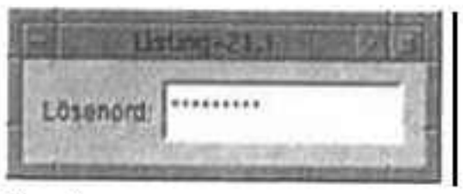


图 21-2　口令部件的瑞典语版本。从二进制翻译文件 21lst01_se.qm 得到翻译

你已经完成了第一个国际化的 Qt 程序！尽管这个例子非常简单，但是无论调用多少次 tr()函数，与这里所用的技术是相同的。

21.4　处理日期和时间值

严格讲，日期和时间值不构成国际化问题。但是，在这个星球上的不同位置，它们的格式也不相同，因此，在本书中将它们看作国际化问题。

Qt 对日期和时间值的支持由 3 个类组成：QDate、QTime 和 QDateTime。所有 3 个类都定义在 qdatetime.h 文件中。

21.4.1　使用 QDate 类处理日期值

到目前为止，QDate::currentDate()是 QDate 类中最流行的函数，它返回当前日期。在 currentDate()函数返回当前日期之后，常用 QDate::toString()函数将日期转换为 QString。程序清单 21-3 给出一个这方面的简单例子。

程序清单 21-3　　一个显示当前日期的部件

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qdatetime.h>
 4: #include <qlabel.h>
 5:
 6: class MyWidget : public QWidget
 7: {
 8: public:
 9:         MyWidget();
10: private:
11:     QLabel *label;
12: };
13:
14: MyWidget::MyWidget()
15: {
```

```
16:     resize( 150, 50 );
17:
18:     //Get the current date and store it
19:     //in date:
20:     QDate date = QDate::currentDate();
21:
22:     //Use the QDate::toString() function to make
23:     //a string out of the date and show it in a
24:     //QLabel:
25:     label = new QLabel( date.toString(), this );
26:     label->setGeometry( 0, 0, 150, 50 );
27:     label->setAlignment( AlignCenter );
28: }
29:
30: main( int argc, char **argv )
31: {
32:     QApplication a( argc, argv );
33:     MyWidget w;
34:     a.setMainWidget( &w );
35:     w.show();
36:     return a.exec();
37: }
```

这个例子创建一个 QDate 对象，并使用 QDate::currentDate()函数定义这个新对象（实际上 QDate::currentDate()返回一个表示当前日期的 QDate 对象）。之后，调用 QDate::toString()函数将日期转换为日期字符串，并将它插入到 QLabel 对象中。该例子执行结果如图 21-3 所示。



图 21-3　一个显示当前日期的小部件

表 21-1 列出其他可能对你有用的 QDate 成员函数。

表 21-1　　QDate 成员函数

函　数	描　述
QDate::year()	返回表示当前年份的整数
QDate::month()	返回表示当前月份的整数（1 为元月，12 为 12 月）
QDate::day()	返回一个整数表示为月中的第几天（1～31）
QDate::dayOfYear()	返回一个整数表示为年中的第几天（1～365）
QDate::dayOfMonth()	返回一个整数表示月中天数（28、29、30、31）
QDate::daysInYear()	返回一个整数表示年中天数（365 或 366）
QDate::monthName()	这个函数带一个整数（1-12）做参数，并将月份名称返回到 QString 对象中
QDate::dayName()	这个函数带一个整数（1-7）做参数，并以字符串形式返回周几名称
QDate::daysTo()	这个函数带一个 QDate 对象做参数，并返回当前日期到参数所指定日期间相隔天数

QDate::currentDate()函数使用日/月/年标准。但是，使用表 21-1 中所描述的函数能够很

容易地实现你所要的标准。

QDate 的另一个大的功能是它能够与==、!=、<、>、<=和>=操作符一起使用。这使多个 QDate 对象的比较变得更加简单。

21.4.2　使用 QTime 类处理时间值

QTime 的构造与 QDate 非常类似。但是，它用于处理时间值，而不是日期值。使用 QTime::currentTime()函数能够检索当前时间，并且可以使用 QTime::toString()函数将时间转换为一个字符串。程序清单 21-4 给出一个这方面的简单例子：

程序清单 21-4　　一个显示当前时间的部件

```
 1: #include <qapplication.h>
 2: #include <qwidget.h>
 3: #include <qdatetime.h>
 4: #include <qlabel.h>
 5:
 6: class MyWidget : public QWidget
 7: {
 8: public:
 9:         MyWidget();
10: private:
11:     QLabel *label;
12: };
13:
14: MyWidget::MyWidget()
15: {
16:     resize( 150, 50 );
17:
18:     //Get the current date and store it
19:     //in time:
20:     QTime time = QTime::currentTime();
21:
22:     //Use the QTime::toString() function to make
23:     //a string out of the time and show it in a
24:     //QLabel:
25:     label = new QLabel( time.toString(), this );
26:     label->setGeometry( 0, 0, 150, 50 );
27:     label->setAlignment( AlignCenter );
28: }
29:
30: main( int argc, char **argv )
31: {
32:     QApplication a( argc, argv );
33:     MyWidget w;
34:     a.setMainWidget( &w );
35:     w.show();
36:     return a.exec();
37: }
```

这里，创建一个 QTime 对象，并使用 QTime::currentTime()函数设置当前时间（第 20 行）。之后，调用 QTime::toString()函数（第 25 行）将它显示在一个 QLabel 对象之中（如图 21-4 所示）。

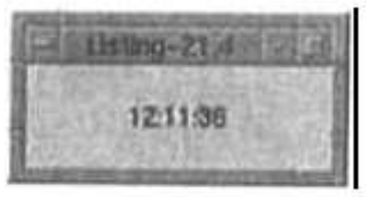


图 21-4　一个显示当前时间的小部件

与 QDate 相同，也能够使用各种不同的比较操作符来比较不同的 QTime 对象。QTime 具有一些成员函数，使用它们能够对 QTime 对象执行各种操作。这些函数如表 21-2 所示。

表 21-2　　QTime 成员函数

函　数	描　述
QTime::hour()	返回表示当前小时值的整数（1～23）
QTime::minute()	返回表示当前分钟值的整数（0～59）
QTime::second()	返回表示当前秒值的整数（0～59）
QTime::msec ()	返回一个整数表示毫秒值（0～999）。在显示当前时间值时很少用它，只有当需要非常准确地测量时间值时才会用到它
QTime::addSecs()	它带一个整数参数，将时间值加上这个秒数后返回新的时间值
QTime::secsTo()	带一个 QTime 对象做参数，并返回另一个 QTime 对象，它表示参数时间值与这个对象时间值之间的时间差
QTime::addMSecs()	带一个整数做参数，参数值表示要加到一个时间值上的毫秒数，之后以 QTimr 对象格式返回新的时间值
QTime::msecsTo()	以 QTime 对象做参数，并返回另外一个 QTime 对象，它表示参数指定时间和这个对象之间的时间差
QTime::start()	如果需要使用 QTime 对象做定时器，调用这个函数启动定时器。之后调用 QTime::elapsed()函数可以检查已经过去了多长时间，并可调用 QTime::restart()重新启动定时器。

QTime 是一个非常有用的类。实际上，由于其使用简单而被广泛用在非 Qt 程序中。无论是显示当前时间值，还是测量某个任务所花费的时间，都建议你使用它。

21.4.3　使用 QDateTime 类处理日期时间值组合

Qt 的 QDateTime 类将 QTime 和 QDate 组合成一个类。这个类常用于显示当前日期和时间。其构造与 QTime 和 QDate 非常类似。但是，它自己能够读取当前日期和时间，这需要使用 QDate 和 QTime 类才能够实现。程序清单 21-5 给出一个例子。

程序清单 21-5　　一个显示当前日期和时间的部件

```
1: #include <qapplication.h>
2: #include <qwidget.h>
3: #include <qdatetime.h>
4: #include <qlabel.h>
5:
6: class MyWidget : public QWidget
7: {
8: public:
9:         MyWidget();
10: private:
11:     QLabel *label;
12: };
13:
14: MyWidget::MyWidget()
15: {
16:     resize( 150, 50 );
```

```
17:
18:     //Create a QDateTime object:
19:     QDateTime datetime;
20:
21:     //Set the date the current date:
22:     datetime.setDate( QDate::currentDate() );
23:
24:     //Set the time to the current time:
25:     datetime.setTime( QTime::currentTime() );
26:
27:     //Use the QDateTime::toString() function to make
28:     //a string out of the date and time and show it
29:     //in a QLabel object:
30:     label = new QLabel( datetime.toString(), this );
31:     label->setGeometry( 0, 0, 150, 50 );
32:     label->setAlignment( AlignCenter );
33: }
34:
35: main( int argc, char **argv )
36: {
37:     QApplication a( argc, argv );
38:     MyWidget w;
39:     a.setMainWidget( &w );
40:     w.show();
41:     return a.exec();
42: }
```

这里，使用 QDateTime::setDate()（第 22 行）和 QDateTime::setTime()（第 25 行）函数设置日期和时间。这些函数用 QDate 和 QTime 对象做参数。在这里，使用 QDate::currentDate()（第 22 行）和 QTime::currentTime()（第 25 行）函数检索对象的当前日期和时间。之后，使用 QDateTime::toString()函数（第 30 行）将当前日期和时间插入到一个 QLabel 对象。

QDateTime 也有一些有用的成员函数，这些函数如表 21-3 所示。

表 21-3　　QDateTime 成员函数

函　数	描　述
QDateTime::date()	返回一个 QDate 对象，它表示 QDateTime 对象中的日期部分
QDateTime::time()	返回一个 QTime 对象，它表示 QDateTime 对象中的时间部分
QDateTime::setTime_t()	带一个 uint 类型参数，它表示自 1970 年 1 月 1 日 00:00:00 后所度过的秒数。当与采用这种方法的标准 C/C++日期和时间函数一起使用 QDateTime 时，这个函数就显得有用
QDateTime::addDays ()	带一个整数参数，参数表示要加到 QDateTime 对象上的天数，之后返回新的 QDateTime 对象
QDateTime::addSecs()	带一个整数参数，参数表示要加到 QDateTime 对象上的秒数，之后返回新的 QDateTime 对象
QDateTime::daysTo()	用 QDateTime 对象做参数，并返回一个整数，返回值表示参数和这个 QDateTime 对象之间相差天数
QDateTime::secsTo()	与前一个函数相同，但这个函数是用秒数而不是天数

你所学到的知识应该能够满足处理日期和时间值方面的所有需要。但是，你应该了解这一节所介绍的 3 个类都具有构造函数，除成员函数外，也可以使用构造函数设置对象的日期和/或时间。QTime 构造函数带 2 到 4 个参数，表示时、分、秒和毫秒。然而，后 2 个参数具有缺省值，因此，它们可以被省略。QDate 构造函数有 3 个参数，表示年、月和日。QDateTime 函数有两个重载构造函数。一个带 QDate 对象做参数，另一个带 QDate 和 QTime

对象做参数。选择使用哪个构造函数依赖于你是只需要设置日期还是需要同时设置日期和时间。

21.5　小　结

Qt 程序能够使用多种语言这一功能必将改善应用程序的使用——对以非英语语言所创建的程序更是如此。

当然，多数人不具备将他们的程序翻译为两种（也可能为 3 种以上）语言的能力。但是，通过使用英语作为程序的缺省语言，并从源代码创建一个翻译文件，然后，你可以将这个翻译文件发送给熟悉那种语言的翻译人员，让他或她在翻译文件中进行填空，之后将它发回给你。此外，程序中所用的很多标准术语很可能在其他一些程序中已经翻译为你所需要的语言。

你也应该考虑将程序翻译为哪种语言。首先应该是使用最广的语言，如英语、德语、法语、西班牙语、葡萄牙语、汉语和日语。当然，你也要考虑哪些人最需要翻译。

由于 Qt 的日期和时间类创建得很好并且容易使用，因此关于日期和时间不需要多讲。与 time.h 中定义的标准日期和时间函数相比，它们能够你节省大量的工作量。

21.6　问题与答案

问：我的程序拒绝使用我所创建的翻译文件，这是什么原因？

答：有很多原因。但是，最大可能是你忘了执行 Q_OBJECT 宏和对类声明使用 moc。其他错误原因包括忘记使用 msg2qm 实用程序创建二进制翻译文件和在程序修改后忘记修改翻译文件。

问：有什么方法能够在翻译文件中搜索一个指定项？

答：有，使用 QTranslator::find()函数能够很容易地实现这一点。它带两个参数，第一个参数表示范围（通常为进行翻译的类名），第二个表示键（需要翻译的字符串）。

问：当使用当前时间时，时间将自动调整为程序运行时区的当地时间吗？

答：是的，这依赖于计算机的系统时钟。因此，如果它被设置为当地时间，当调用 QTime::getCurrent()时 QTime 将得到当地时间。这也意味着如果你的系统时钟设置是错误的，QTime::getCurrent()将返回错误的时间。

21.7　作　业

完成下面的问题和练习将有助于你牢记这一学时中所学的怎样国际化 Qt 程序和怎样使用 Qt 类处理日期和时间值。

21.7.1　测验

1. 什么是用户空间文本？

2. 为什么应该对所有的用户空间文本使用 QString？
3. 什么时候需要使用 tr()函数？
4. findtr 实用程序的用途是什么？
5. 什么时候需要实用 msg2qm 实用程序？
6. 什么时候需要实用 mergetr 实用程序？
7. 哪个类在程序中执行实际翻译操作？
8. QApplication::installTranslator()函数做什么？
9. 为什么需要调用 Q_OBJECT 宏，并对使用翻译的类使用元对象编译器？
10. QDate 用于处理什么？
11. QTime 用于处理什么？
12. 什么时候应该考虑使用 QDateTime 类？

21.7.2　练习

1. 基于 QMainWindow 编写一个标准程序。向它添加标准菜单（File、Edit 和 Help）。再添加一个具有几个按钮的工具栏和一个状态栏，当用户鼠标指针指向按钮时，状态栏显示其描述信息。之后，将整个程序翻译为另外一种语言。程序的一个有趣功能是当程序启动后使用户能够通过一个对话框选择语言。记住必须遵守在“创建翻译文件”和“在程序中实现翻译功能”两节中所介绍的所有步骤。
2. 如前所述，QDate 函数使用日/月/年格式显示日期。但是，很多人使用月/日/年格式。因此，使用 QDate 的成员函数以月/日/年格式显示一个日期。

第 22 学时　可移植性

如果在一个平台下所创建的程序或库能够很容易地被转换到其他平台，它就被认为是可移植的。正如通过将程序翻译为多种语言能够使它被更大范围内的人所使用一样，使程序适用于多个平台也能够使它被更多的人所使用。因此，除功能之外，移植性应该是应用程序开发过程中最优先考虑的。

Qt 的最大优点之一就是其移植性。在编写本书时，能够很容易地使用各种编译器在 16 种不同的 UNIX 版本及其兼容产品上编译 Qt。当然，使用商用版本的 Qt，你的程序也能够在 Microsoft Windows 上编译。事实上，这意味着能够使你的程序在地球上大多数计算机上运行。

但是，为了使 Qt 程序具有移植性，应该采取一些措施。这是这一学时中的话题。这有助于你编写出好的、可移植的应用程序，使它们应用在更大范围内。如果你正在执行一项商业计划，这或许能够帮助你赚到更多的钱。

尽管在多数情况下希望具有移植能力，但是在一些情况下不必强求移植性。例如，如果你在创建一个用于编译 Linux 内核的 GUI 配置工具，移植性就一点也不重要，因为这个程序很可能从不会用于 Linux 以外的平台。如果你正在为 Windows 下的设备驱动程序开发一个 GUI 安装工具，也就根本不需要考虑移植性。但是，在多数情况下，移植性是很重要的，因此，你很可能将发现这一学时是很令人感兴趣的。

22.1　编写可移植的 Qt 应用程序

如前所述，Qt 具有极好的可移植性，但是，为了使 Qt 程序具有移植性需要程序员做一些工作。在这一节，将介绍保持 Qt 程序移植能力方面的通用信息。

22.1.1　使用 Qt 类

编写 Qt 应用程序时使用 Qt 类和函数对你来说可能是很自然的。但是，当不需要使用与 GUI 相关的函数时，你很可能在程序中使用其他库。在程序中使用其他库将使程序变得更难（尽管不是不可能）移植。最坏的情况是你所用的这个库只支持当前开发环境，因此，这使你的程序除了只能运行在最初平台外，不可能再运行在其他平台下。你始终应该牢记，Qt 组成不仅有 GUI 部件，而且还有一些函数，这些函数用于实现其他程序设计任务，如访问文件系统等。

一、使用 Qt 流函数

如你在第 12 学时“处理文件和目录”中所学到的，Qt 对文本流提供广泛的支持。在多数情况下，这些流函数也足以满足你的需要。如果你想保持程序的移植性，使用 Qt 流函数就显得非常重要。

尽管在 C++实现中存在标准 C++流类，定义在 iostream.h 中，并且使用它们也能够保持移植功能，但是，一些小的差别使不同实现之间或多或少不能够完全相互兼容。这些小的差别实际上是不同实现中的内置类型（如 int、long 和 double）长度间的差别，例如，int 在一个系统上可能为两字节，而在另一个系统上可能为 4 字节。因为这些类型被标准流类所使用，这些小的差别可能在实际使用标准流的程序中产生大问题，它们可能使在一个平台下的有效程序而在另一个平台下变得无效。因此，强烈建议你坚持使用 Qt 流类，它们能够帮助你避免很多问题。

如果你想为程序创建特殊的文件类型，QDataStream 类就是一个很好的类。尽管使用标准 C++类也能够实现这一功能，但是，QDataStream 在 Qt 支持的所有平台下保持完全一致，因此保持你程序的可移植性。例如，如果你正在编写一个字处理程序，使用 QDataStream 类能够很容易地创建一个用于存储文档的特殊文件格式。因此，如果你在一个平台下保存一个文档，同一文件可以毫无问题地在所支持的所有其他平台（Qt 支持的所有平台）下打开。

二、使用 Qt 类处理文件和目录

QFile 和 QDir 类是用于处理文件和目录的 Qt 类。这些在第 12 学时中已经讨论过。如果使用标准 C 或 C++库，你知道它们中也存在这些结构。但是，这些结构存在与标准流类同样的问题，因此，为了保持程序的可移植性，建议你使用 QFile 和 QDir 类。QFile 和 QDir 类比标准 C 和 C++标准结构更容易使用。

三、用 QPaint 绘图

在第 19 学时“使用 Qt 的 OpenGL 类”中简单讨论了在 Qt 程序中实现 OpenGL 函数之前应该考虑再三。这里接着讨论这个问题。使用 QPainter 而不是一个全新的绘图库能够使用户更加容易地使用它，并且可以确保使程序中的绘图功能能够在 Qt 所支持的所有平台下正常运行。

四、用 QPrinter 打印

使用 Qt 的 QPrinter 类是目前在程序中实现打印功能的一种最简单方法。如果你正在考虑不使用这个类实现打印功能，则应该尽快抛弃这一想法！QPrinter 不仅容易使用，而且它能够确保程序的打印功能在所有平台下都正常工作（如果一台打印机被正确安装和配置，就能正常打印）。

五、用 Qt 处理套接字

Qt 套接字类能够用于处理各种输入源，如网络连接。Qt 套接字支持由 3 个类组成：

QSocket、QSocketAddress 和 QSocketNotifier。用这 3 个类能够创建与平台无关的套接字通信系统。在编写本书时，Qt 套接字支持还没有完全实现。但是，它在不久的将来就会被实现。想象一下你自己的与平台无关的 Internet 浏览器！

在编写本书时，Qt 套接字库还不是标准 Qt 库的一部分。当你阅读本书时是否仍是这种状况还很难说。但是，很明显，当你编译使用 Qt 套接字类的程序时，如果编译器报告关于这 3 个套接字类的未定义符号错误，就说明这个库还没有成为标准 Qt 库的一部分。然而，修正这一点非常简单，按下面的方法执行：

```
# cd $QTDIR/extensions/network/src
# make
```

通过运行 make，在$QTDIR/lib 下安装和编译 libnetwork.a 库（一个静态库）。之后，当使用套接字类时，与这个库进行链接。

22.1.2　遵守 POSIX 标准

POSIX 是一个操作系统标准，它是 IEEE 可移植操作系统计算接口（Portable Operating System Interface for Computing）的缩写。POSIX.1 为 POSIX 标准定义部分，它定义访问系统相关功能可以使用的函数。如果使用遵守这一标准的函数，你就能够用非 Qt 类和函数并仍保持程序的可移植性。

所有有名的 UNIX 系统和 Windows NT 都遵守 POSIX 标准。如果你需要 Qt 未提供的函数，你就必须使用外部函数或类。但是，如果非 Qt 函数或类遵守 POSIX 标准，就仍能够保证你的程序是可移植的。

如果需要深入学习 POSIX，并了解哪些函数是 POSIX 标准的一部分，可以拨打 IEEE 电话 732-981-1391 订购 POSIX 标准和草案（在美国，也可以拨打免费电话 1-800-678-IEEE）。

22.1.3　隔离平台相关的调用

有时，一个问题的唯一解决办法只能是调用与平台相关的甚至可能是不遵守 POSIX 标准的函数。在这些情况下，你必须将程序中的某些部分编写两个或多个版本，以使它们能够在多个平台下使用。

不要将平台相关的代码扩展到整个源代码，这一点非常重要。这将使移植过程更加困难和耗时。如果你将它扩展到整个源代码，则必须搜索整个源代码以查找平台相关的调用，并在这里或那里做一些小的修改。相反，如果将与平台相关的源代码隔离到一个或几个类或文件之中，移植将变得更加容易，因为这时你知道代码中的哪一部分需要修改。

例如，可以将与平台相关的代码隔离到 PlatformSpecificClass 类，并为每一个所支持的平台创建一个这样的类。之后，用一个全局头文件 PlatformSpecificClass.h 存储 PlatformSpecificClass 类声明，并为每一个平台创建一个 CPP 文件，它们存储与平台相关的类定义。如果程序支持 Linux 和 Windows NT，则只需创建两个 CPP 文件——PlatformSpecificClass-Linux.cpp 和 PlatformSpecificClass-NT.cpp，并将其中的一个拷贝到 PlatformSpecificClass.cpp 中，这依赖于是在 Linux 还是在 Windows NT 平台下编译。

只要遵循这些简单的隔离规则，移植一个使用平台相关代码的应用程序将变得非常简单（尽管这不会像只使用 Qt 类那样简单）。



实现这一功能的另外一种方法是创建两个具有相同名称的类，将它们的类声明放置在不同的文件中，之后由预处理器决定包含哪个头文件。看下面的例子：

```
#ifdef LINUX
#include "platform-linux.h"
#else
#include "platform-NT.h"
#end if
```

要确保只有当在 Linux 平台下编译程序时才定义 LINUX，之后，将这段代码添加到 cpp 文件头部，就能够保证包含正确的头文件。

22.2　不可移植的 Qt 函数

尽管 Qt 开发人员尽可能保证 Qt 的平台无关性，但是，在某些情况下，不可能完全这样做。因此，几个 Qt 方法的功能在 UNIX 和 Windows NT 下有一些差别。尽管这些差别不大，但它们在某些情况下仍可能导致出现问题。然而，如果你了解了这些差别，就能够很容易地绕过它们。表 22-1 列出了 Qt 2.0.2 中在 UNIX 和 Windows NT 下具有不同功能功能的方法，这些函数以字母顺序排列。

表 22-1　　不可移植的 Qt 函数

函　数	解　释
debug()	debug()函数用于向程序源输出调试信息。但是，其功能在 UNIX 和 Windows NT 下有一点差别。在 UNIX 下，给 debug()的参数被输出到 stderr（通常为屏幕）；在 Windows NT 下，它们被传送到调试器。但是，在程序的最终版本中你可能要删除对 debug()的调用，因此，这几乎不可能出现问题
QApplication::flushX()	在 X11 系统（UNIX）下，对该函数的调用将刷新 X 事件队列。但是，它在 Windows 下无任何作用
QApplication::setColorSpec()	该函数的功能依赖于它是在 UNIX 还是在 Windows 下调用　在第 9 学时“创建简单图形”中的“使用颜色”一节中已经讲述过这一内容
QApplication::setDoubleClick-Interval()	使用这个函数，能够设置一次鼠标双击中两次鼠标点击间的最大时间间隔，单位为毫秒。在 Windows 下，这个函数设置所有窗口中的双击值，而在 UNIX 下，它所设置的双击值只影响你的 Qt 程序
QApplication::setMainWidget()	在 UNIX 系统下，调用这个函数将按照-geometry 命令选项设置窗口的位置和大小。但在 Windows 下就不是这样
QApplication::syncX()	在 UNIX 系统下，这个函数处理所有的未完成事件，并刷新所有队列。但是，在 Windows 下就不需要这样做，因此，在该平台下该函数没有什么用处
QDir::convertSeparators()	在 Windows 下，这个函数用于将路径中的斜杠(/)转换为反斜杠(\)。例如，QDir::convertSeparators("c:/windows/system")将返回 c:\windows\system。在 UNIX 下，这个函数无用。但是，QDir::convertSeparators()应当被看作为一个优点而不是一个问题
QDir::setFilter()	这个函数用于设置文件过滤器。这意味着你能够决定 QDir::entry()和 QDir::entryInfoList()应该返回哪种类型的文件。这样做需要向这个函数传递一个或多个值。但是，其中的两个值——Modified（列出被修改文件）和 System（列出系统文件）——在 UNIX 下毫无作用。此外，在 UNIX 下，Hidden 将导致列出以点(.)开头的文件
QFile::open()	传递给这个函数的参数值 IO_Translate 将启动回车和换行转换功能。这在 UNIX 下不需要，因此，IO_Translate 无任何作用

续表

函　数	解　释
QWidget::setSizeIncrement()	传递给这个函数的两个整数参数定义窗口在水平和垂直方向上调整的最小像素数。换句话说，能够定义窗口的调整步骤是多大。它在 Windows 下无任何作用。一些 X11 下的 Window Manager 也忽略该函数
QWidget::winId()	这个函数返回部件的窗口标识号（类型为 wid）。尽管在 UNIX 和 Windows 下都完全能够使用这个函数，但是由于返回值不同，因此不可移植

如果你想使程序能够移植，就应该避免使用这些函数。如果必须使用这些函数——例如，QApplication::setMainWidget()就很难避免——你应该知道 Windows 和 UNIX 实现是完全不兼容的，因此，你可能要采取一些进一步的措施解决这个问题。所幸的是，表 22-1 中的函数没有一个会导致破坏性问题。更重要的是，如果出现一个问题，这些函数通常也很容易被解决。

22.3　用 tmake 实用程序构造可移植项目

你很可能已经注意到，在一个程序的源分发程序中总有一个或多个文件叫做 makefile。这些文件存储程序编译信息。该信息包括使用哪个编译器、链接哪个库以及以什么样的顺序编译源文件，等等。之后，当你在一个包含编译文件的目录下运行 make 实用程序时，make 将读取编译文件，并按照编译文件中的定义编译文件。采用这种方法，任何人想编译源程序时，只要输入 make 后按回车键即可。

可以用不同的方法创建编译文件。然而，尽管你肯定能够以手工方式编写所有的编译文件，但是使用一些工具来创建编译文件将会更加方便。如果你曾经在 UNIX 下编译过应用程序，你很可能熟悉 configure 脚本程序，它常用于读取各种类型的系统信息，并基于该信息创建一个或多个编译文件。

然而，这只适用于 UNIX 平台。当创建一个 Qt 这样的交叉平台库时，你需要使用一些能够同时运行在 UNIX 和 Windows 下，并能够为这两个平台产生编译文件的工具。幸运的是，Troll Tech 已经创建了一个这样的工具，叫做 tmake。这个工具能够根据它当前所运行的平台为 UNIX 或 Windows 创建编译文件。事实上，这是一个特殊工具，它能够为你节省大量令人厌烦的工作。下面是 Troll Tech 对该工具的描述：

tmake 自动而高效地管理编译文件，它使你将宝贵的时间花费在编写代码上，而不是在编译文件上。

22.3.1　获取和安装 tmake

tmake 是一个自由软件，可以从 Troll Tech 的 FTP 站点下载（ftp://ftp.troll.no/freebies/tmake/）。这里，你将找到该工具的两个版本——一个用于 Windows，一个用于 UNIX。Windows 版本以.zip（它是一个压缩文件）结尾，UNIX 版本以.tar.gz 结尾。在编写本书时，tmake 的最新版本为 1.3。

当下载完一个 tmake 分发程序后，需要解开这个文件。在 Windows 下，使用 WinZip(www.winzip.com)或 pkzip(www.pkware.com)能够很容易地实现该操作。如果在 UNIX 下，在包含你所下载的文件目录下执行下面命令：

```
# tar xvfz tmake-1.3.tar.gz
```

这将创建一个叫做 tmake 的子目录，所有 tmake 分发程序都在这个目录下。很可能你需要将它移动到/usr/local/qt 或其他更合适的目录下。为了更方便地执行 tmake，你应该在 PATH 变量所包含的目录（如/usr/bin）下创建一个到/usr/local/qt/tmake/bin/tmake 的符号连接。如：

```
# ln -s /usr/local/qt/tmake/bin/tmake /usr/bin/tmake
```

在 Windows 下，你很可能需要将 c:\tmake\bin 目录添加到你的变量中，以便在每次使用 tmake 时不必输入其全部路径。

在 UNIX 和 Windows 下，都必须设置 TMAKEPATH 变量，它指向系统中 tmake 模板文件的存储目录。模板文件记录系统各种信息，tmake 使用该信息创建编译文件。因此，如果在 UNIX 下，使用 Bourne 外壳，编译器为 g++，则应将下面内容添加到你的启动文件中：

```
TMAKEPATH=/usr/local/qt/tmake/lib/linux-g++
export TMAKEPATH
```

如果使用 C 外壳，则应使用下面内容：

```
setenv TMAKEPATH /usr/local/qt/tmake/lib/linux-g++
```

如果你使用 Windows 和 Visual C++编译器，则应将在 autoexec.bat 中设置 TMAKEPATH 和 PATH 变量：

```
set TMAKEPATH=c:\tmake\lib\win32-msvc
set PATH=%PATH%;c:\tmake\bin
```

需要列出模板文件中预定义的所有平台和编译器时，只需在 tmake 安装下的/lib 目录中运行 dir 命令即可。

当你按照这些步骤中的一种来设置 TMAKEPATH 变量，并正确安装 tmake 分发程序后，所有工作就设置完成。



注意，当将一些内容添加到启动文件时，如/etc/profile，需要注销后重新登录才能激活所做的修改。如果你不想注销后重新登录，可以对文件使用 source 命令。例如：

```
source /etc/profile
```

这将读取文件/etc/profile 并执行其中的所有命令，包括 TMAKEPATH 变量设置行。



tmake 是一个用 perl 编写的脚本程序。因此，如果使用 UNIX，你需要安装一个 Perl 解释器，perl，才能使用 tmake。所幸的是，perl 常被现在的 UNIX 系统缺省安装。如果它没有安装，它很可能就包含在你的 Linux/UNIX 分发程序中。

22.3.2　用 tmake 创建编译文件

当在系统中正确设置 tmake 后，你就可以开始使用它。在这一节，将介绍 tmake 的最普

通用途。其完整描述可参看 HTML 文档 tmake.html（用户指南）和 tmake-ref.html（参考文档）。这两个文件都位于 tmake 分发程序下的 doc 子目录内。

当 tmake 为你创建编译文件时，它基于在 TMAKEPATH 指定目录中所查找到的模板文件并依赖于你需要创建的项目文件（以.pro 结尾）。创建项目文件非常简单，通常也花费不了很长时间。这里是一个例子：

```
CONFIG   =   qt release
HEADERS  =   MyWidget.h
SOURCES  =   MyWidget.cpp main.cpp
TARGET   =   MyWidget
```

在第 1 行，设置 CONFIG 变量。这个变量有几个预定义值，它们决定项目类型。在这里，选项设置为 qt 和 release。这告诉 tmake 该项目为一个 Qt 项目，并且应该将它看作发行版本而不是调试版本。然而，缺省时（如果没有设置 CONFIG 变量）这两个选项都被设置。这里包含它只是作为一个例子。在 CONFIG 变量中可以输入的全部选项列表如表 22-2 所示。

表 22-2　　CONFIG 选项

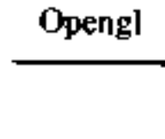
函　数	解　释
qt	该选项告诉 tmake 项目为一个 Qt 项目。当设置该选项时，将产生和包含所有需要的 MOC 文件。这是缺省设置
release	该选项告诉 tmake 把项目作为为一个发行版本。这意味着将不包含任何调试符号，并优化编译二进制文件。这也是缺省设置
debug	该选项告诉 tmake 把项目作为一个调试版本，因此其中将包含调试符号。只有当你需要编译一个二进制文件用于调试时，才应该设置该选项
warn_on	如果设置该选项，tmake 将尽可能多地显示编译期间的警告信息。然而，当 CONFIG 变量未被设置时，这是缺省设置
warn_off	该选项告诉 tmake 尽可能少地显示编译期间的警告信息。但是，建议你不要这样做，因为它可能隐藏一些严重问题
Opengl	如果项目使用 Qt 的 OpenGL 扩展则设置该选项

接下来，将 HEADERS 变量设置为 MyWidget.h。在这里，你应该列出程序中的所有头文件。尽管在这个例子中只有一个。SOURCE 变量应该包含所有的源文件（CPP 文件）。在这个例子中，有两个源文件——一个包含 Qt 类定义，一个包含 main()函数。最后一个变量，TARGET，定义可执行输出文件（实际程序）。这里，TARGET 被设置为 MyWidget。

现在，你已经将 TMAKEPATH 变量设置到系统中的相应目录，并且也有一个简单（但很有用）的项目文件。这样就可以使 tmake 产生编译文件。为此，调用下面命令：

```
tmake MyWidget.pro -o Makefile
```

如果所有设置都正确的话，tmake 将产生一个完全可用的编译文件（叫做 Makefile）。



> 注意，如果你想执行 make 而不带任何参数，就必须将编译文件命名为 Makefile。

如果将 TMAKEPATH 设置为使用系统中的 Linux/g++模板文件，所产生的编译文件将如程序清单 22-1 所示。

程序清单 22-1　　tmake 产生的一个编译文件

```
#############################################################################
# Makefile for building MyWidget
# Generated by tmake at 09:02, 2000/01/29
#     Project: MyWidget
#    Template: app
#############################################################################

####### Compiler, tools and options

CC    =    gcc
CXX    =    g++
CFLAGS    =    -pipe -O2 -DNO_DEBUG
CXXFLAGS=    -pipe -O2 -DNO_DEBUG
INCPATH    =    -I$(QTDIR)/include
LINK    =    g++
LFLAGS    =
LIBS    =    -L$(QTDIR)/lib -lqt -L/usr/X11R6/lib -lXext -lX11 -lm
MOC    =    moc

TAR    =    tar -cf
GZIP    =    gzip -9f

####### Files

HEADERS =    MyWidget.h
SOURCES =    MyWidget.cpp \
        main.cpp
OBJECTS =    MyWidget.o \
        main.o
SRCMOC    =    moc_MyWidget.cpp
OBJMOC    =    moc_MyWidget.o
DIST    =
TARGET    =    MyWidget

####### Implicit rules

.SUFFIXES: .cpp .cxx .cc .C .c

.cpp.o:
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

.cxx.o:
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

.cc.o:
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

.C.o:
    $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

.c.o:
    $(CC) -c $(CFLAGS) $(INCPATH) -o $@ $<

####### Build rules

all: $(TARGET)
```

```
$(TARGET): $(OBJECTS) $(OBJMOC)
    $(LINK) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(OBJMOC) $(LIBS)

moc: $(SRCMOC)

tmake: Makefile

Makefile: MyWidget.pro
    tmake MyWidget.pro -o Makefile

dist:
    $(TAR) MyWidget.tar MyWidget.pro $(SOURCES) $(HEADERS) $(DIST)
    $(GZIP) MyWidget.tar

clean:
    -rm -f $(OBJECTS) $(OBJMOC) $(SRCMOC) $(TARGET)
    -rm -f *- core

####### Compile

MyWidget.o: MyWidget.cpp \
        MyWidget.h

main.o: main.cpp \
        MyWidget.h

moc_MyWidget.o: moc_MyWidget.cpp \
        MyWidget.h

moc_MyWidget.cpp: MyWidget.h
    $(MOC) MyWidget.h -o moc_MyWidget.cpp
```

这不是一个注释性内容，因此你在这里的清单中不能得到全部解释。但是，编译文件所使用的语法非常简单，因此，没有解释你也应该能够理解这个清单。最有趣的部分是最后几行，它们确保正确产生和包含 MOC 文件。

注意，如果使用 tmake，你不必手工产生和包含 MOC 文件，相反，tmake 会处理它。在这里，你只需在 MyWidget.cpp 和 main.cpp 中包含 MyWidget.h 文件，并且要确保在 MyWidget.h 中包含对 Q_OBJECT 的调用。

当编译文件产生后，开始编译：

```
# make
```

之后，你将看到以下输出内容：

```
g++ -c -pipe -02 -DNO_DEBUG -I/usr/local/qt/include -o MyWidget.o MyWidget.cpp
g++ -c -pipe -02 -DNO_DEBUG -I/usr/local/qt/include -o main.o main.cpp
moc MyWidget.h -o moc_MyWidget.cpp
g++ -c -pipe -02 -DNO_DEBUG -I/usr/local/qt/include
-o moc_MyWidget.o moc_MyWidget.cpp
g++ -o MyWidget MyWidget.o main.o moc_MyWidget.o -L/usr/local/qt/lib -lqt
-L/usr/X11R6/lib -lXext -lX11 -lm
```

注意，在第 3 行产生一个 MOC 文件，在第 4 行对它进行编译。在第 5 行和第 6 行，3

个目标文件（包括从 MOC 文件产生的目标文件）被链接到一起，形成一个可执行二进制文件，叫做 MyWidget。



为了编译这个项目，需要在当前目录中有 MyWidget.cpp、MyWidget.h 和 main.cpp 文件，并且在它们之中要有一些真正的 C++代码。但是，你也可以使用前面的一个例子。

你已经完成了用 tmake 创建第一个编译文件，在这一节开始已经讲过，在 tmake 安装下的 doc 子目录中的两个 HTML 文档 tmake.html 和 tmake-ref.html，它们分别包含一个完整的用户指南和一些参考文档，查阅它们可以进一步获得关于 tmake 的相关信息。

22.4　用 progen 产生项目文件

如果你正在开发一个具有多个文件的大型 Qt 项目，在项目文件中手工列出所有文件将是非常烦琐的。所幸的是，Troll Tech 已经开发了一个解决这一问题的工具，它叫做 progen（Project Generator）。与 tmake 一样，progen 也是一个 Perl 脚本程序，并且包含在 tmake 分发程序中。如果你在使用 UNIX，而没有将/usr/local/qt/tmake/bin 目录添加到 PATH 变量中，就需要像处理 tmake 一样创建一个到/usr/bin 目录中 progen 的符号连接。其创建方法为：

```
# ln -s /usr/local/qt/tmake/bin/progen /usr/bin/progen
```

采用这种方法，就能够在系统中的任何位置不输入 progen 的完整路径而执行它。

progen 的作用是搜索指定的一组源文件，并根据它所查找到的这些文件信息产生项目文件。但是，progen 不可能准确知道你想使用的选项（像需要链接哪些库等），因此，还需要进行手工编辑。

这里是一个例子，它使用前一节中的文件（MyWidget.cpp、MyWidget.h 和 main.cpp）。将这些文件放置到它们自己的目录下，并执行下面命令：

```
# progen -n MyWidget -o MyWidget.pro
```

现在，progen 将在当前目录及其子目录下搜索所有以.cpp 或.h 结尾的文件。之后创建一个项目文件（项目文件所包含的内容依赖于 progen 所查找到的文件）。-n 选项用于定义项目名称（也就是项目文件中 TARGET 变量值）。使用-o 选项定义项目名称。在这个例子中，progen 将创建下面项目文件：

```
TEMPLATE    =    app
CONFIG  =    qt warn_on release
HEADERS =    MyWidget.h
SOURCES =    MyWidget.cpp \
         main.cpp
TARGET  =    MyWidget
```

如你所看到的，progen 查找到 3 个文件，并将它们正确放置到 HEADERS 和 SOURCES 变量中。然而，在这个例子中，因为项目只包含 3 个文件，所以 progen 没有多大帮助。但是，如果项目中包含几百个文件，就可以想象这个工具是多么地有用！



在 progen 所产生的项目文件中的第 1 行，有一个你以前没有看到过的新变量，叫做 TEMPLATE。这个变量定义所使用的模板文件名称。在这里，将使用/usr/local/qt/tmake/lib/linux-g++/app.t 模板文件。这也是缺省值，因此，当你在开发应用程序时，不必定义 TEMPLATE 变量。但是，如果项目不是一个应用程序，则可以使用/usr/local/qt/tmake/lib 每个子目录中的其他文件(而不是 app.t)。例如，如果项目是一个库，则应该使用 lib.t。

在 tmake 用户指南中介绍了 progen 实用程序，因此，你应该查阅它以对这个工具有一个全面的了解。

22.5　小　结

无论是专业开发 Qt 程序，还是只是为了好玩，可能都希望有尽可能多的人能够使用你的软件。如果你是在销售软件赚钱，尽可能保证其移植性就很重要（因为用户越多就意味着能赚更多的钱）。此外，作为一个自由软件程序员，如果你知道有很多人使用你长期辛苦工作所创建的软件，你也会感到非常高兴。

Qt 本身就是可移植的，所以，你不必做一些特殊的工作来保持其移植性。然而，应该了解 Windows 和 UNIX 实现之间的一些小差别。很多这些差别你很可能从没有经历过。但是，如果出现问题，它也很好理解，因此，能够很容易地处理它。这不会超过一两行代码。

很多 Qt 程序员都认为，如果没有 tmake，他们就无法工作，一旦你了解它，你将也会有同样的感觉。为了完全理解 tmake 的优点，你需要开始一个至少由 200 个源文件和头文件组成的项目。之后，使软件至少应用到 5 个不同的 UNIX 平台和 Microsoft Windows。此外，在开发过程中，手工编写所有的编译文件，为源文件目录树中的每个子目录创建一个，并为每个平台创建一个版本。这样，你就会发现，创建所有编译文件所花费的时间将会和编写代码所花费的时间差不多！但是，如果使用 tmake 和 progen 则会立即产生这些文件。

22.6　问题与答案

问：当我执行 tmake 和 progen 时，出现查找不到 perl 错误，这是什么原因？

答：你很可能没有安装 perl。从你的分发程序中安装它，或者从 www.perl.com 站点下载。

问：tmake 实用程序无法工作，它报告未查找到 tmake.conf 文件！

答：如果没有正确设置 TMAKEPATH 变量就会出现该错误。要确保你已经将它添加到启动文件，之后注销后重新登录。另外一种方法，你也可以使用 source 实用程序。如果你使用 bash，则必须使用 export 关键字导出变量。如果使用 C 外壳，则必须使用 setenv 关键字。

问：我正在开发一个具有 GUI 界面的 Linux 驱动程序安装程序，对我来说可移植性很

重要吗？

答：不，它不重要。设备驱动程序总是与平台相关的。人们在其他平台上安装你的设备驱动程序的机会很小。但是，如果你想为 Windows 编写一个类似的程序，使这个 GUI 安装程序具有移植性将是一个很好的想法，因此，它能很容易地适用于 Windows。采用这种方法，能够为你节省许多工作。

问：可以使用 tmake 为我所使用平台以外的其他平台创建编译文件吗？

答：当然可以。只需修改 TMAKEPATH 变量。例如，如果想为 Windows 下的 Borland C++ 创建编译文件，需像下面这样设置 TMAKEPATH：

```
# export TMAKEPATH=/usr/local/qt/tmake/lib/win32-borland
```

之后，使用 tmake 创建编译文件：

```
tmake MyWidget.pro -o Makefile
```

就这样简单！

22.7　作　业

完成下面的问题和练习将有助于你牢记这一学时中所学内容。

22.7.1　测验

1. 使用 Qt 可以编写网络应用程序吗？
2. 什么是 POSIX？
3. 什么是 POSIX.1？
4. Qt 中是否有一些函数在 UNIX 和 Windows 下完全不兼容？
5. 什么是 tmake？
6. 什么是项目文件？
7. 什么是 progen？

22.7.2　练习

1. 将 TMAKEPATH 变量设置到/usr/local/qt/tmake/lib 下的不同子目录。每次修改变量后都创建一个编译文件，查看这些文件的内容有什么差别。
2. 切换到 Qt 安装的 src 目录（例如，/usr/local/qt/src）。在该目录下执行 progen –o MyProject.pro 命令。当 progen 结束后，查看 MyProject.pro 文件。用手工方式怎样编写它？
3. 使用前一个练习中所创建的项目文件使 tmake 产生编译文件。当 tmake 完成后，查看编译文件。想象一下用手工方式来编写它！对这个编译文件做一些小的修改，即可用它来编译整个 Qt 库。

第 23 学时　调试技术

在开发计算机软件时，必须考虑软件中或多或少存在的严重错误（程序员们将它称之为臭虫(bug)）。这些臭虫有很多不同的表现方式。如果在使用 Linux，你很可能经历过程序因为段错误而退出，这通常是由于内存管理错误而引起的。但是，臭虫定义不只是内存问题。实际上，程序中所有有害的东西都可被称为臭虫。计算机程序中的臭虫与化学中的腐蚀有相同的含义，当发生有害的事情时，你都可以使用这个术语（有害的程序功能或者有害的化学反应）。

产生臭虫的一个原因是缺乏对程序员所使用库或函数的理解。例如，刚开始学习时所编写的 Qt 程序存在的问题肯定比现在，学习结束时，所编写的 Qt 程序要多。现在你对 Qt 库工作机制的理解要广，因为你已经知道了一些问题，所以能够很容易地避免它们。因此，避免臭虫最好的方法就是要确实理解你所做的工作。

然而，无论你的经验是多么丰富，总是有一些没有改正或是未被发现的臭虫。所幸的是，Qt 提供一些函数和宏来帮助解决这些问题。它们不能改正臭虫，但是能够帮助你发现它们。毕竟对臭虫来说，最大的问题就是发现它们！

在这一学时，将学习怎样使用 Qt 所提供的查找臭虫功能。也将学习怎样使用调试器来一步一步地调试 Qt 程序。在这一学时的最后一节，将讨论命令行选项，当通过调试器运行 Qt 程序时，这些选项可能有用。

23.1　使用 Qt 调试功能

Qt 带有几个函数，它们使程序调试变得更加容易。在程序中的指定部分调用这些函数，能够很容易地判断这部分是否被执行。例如，使用这些函数能够检测在哪一点 if 语句为 true 或者是一个 while 循环中的代码被执行多少次。这样做，你对程序能够有一个更好的了解，也能够更加容易地避免臭虫。

这一节讨论 3 个函数：qDebug()、qWarning()和 qFatal()。它们均为全局函数，因此不需要包含使用它们的头文件。



所有这 3 个函数的输出都被限制为 512 字节，包括 0 字节（该字节标识字符串结束）。

23.1.1　qDebug()函数

这 3 个函数中最重要并且可能是最常用的函数是 qDebug()。它将调试信息输出到 stdout（屏幕），使用它能够发现程序正在做什么。



> 在 Windows 下，qDebug()将信息输出到调试器，而不是 stdout。

qDebug()带一个或多个参数，第 1 个参数为格式字符串，其余参数应该被插入到格式字符串中。如果你使用过标准 C 库中的 printf()函数，就会知道这怎样工作。看下面例子：

```
qDebug("This is the widget's ID = %x", MyWidget->id( ) );
```

这将使用第 2 个参数，MyWidget->id()，代替第一个参数中的%x。例如，如果 MyWidget 部件的标识号为 3，该行的输出将像下面这样：

```
This is the widget's ID = 3
```

但是，为了使用 qDebug()函数，所有对它的调用都应该放在程序中的适当位置。一般来说，在程序中全局调用 qDebug()没有什么用处。但是，你可以用它检查程序中的某个部分是否或什么时候被执行。例如，你可以像下面这样使用它来检查一个 while 语句体什么时候被执行：

```
while ( YourVariable != Something )
{
    qDebug( "Now the body is executed,
so the developer must be warned!" );
}
```

这里，每当 while 语句为 true 时，Now the body is executed, so the developer must be warned! 文本就被显示到 stdout（Windows 下为调试器）。现在，你能够准确看到什么时候执行程序中的这一部分。如果循环体执行点不是在它应该执行的地方，这能够被你很容易地看出。相反，如果在它应该执行的地方没有执行，这也能够被你很容易地看出。这两种情况都能够帮助你查找臭虫！

23.1.2　qWarning()函数

qWarning()函数的工作与 qDebug()完全一样。但是，从程序设计的角度来看，二者间又有很大的差别。通过使用 qDebug()函数通知某个事件和用 qWarning()函数通知程序错误，这使你能够明显区分程序中的事件信息和错误信息。

如果将 qDebug()用于这两种任务，你将更难判断一个 qDebug()调用所通知的是程序事件还是程序错误。因此，当程序中出现程序错误时，要使用 qWarning()通知。这里是一个例子：

```
QPushButton b1( "Hello!", this );
if( !QPushButton )
{
    qWarning( "The QPushButton object b1 was
not created correctly!" );
```

```
}
```

这里，在第一行创建QPushButton对象，b1。但是，如果出现问题，if语句为true，qWarning()函数将被调用。采用这种方式，当 b1 创建出现问题时就能够通知你。这是一个典型的什么时候应该使用 qWarning()函数的例子。



你应该考虑做的另一件事情是在使用 qWarning()和 qDebug()时应该标识它们的输出，以便在程序运行时你能够很容易地看到是执行了哪个函数。这里是一个例子：

```
qWarning( "WARNING: The QPushButton object
    b1 was not created correctly! " );
```

在输出的开始插入 WARNING 字，你能够很容易地辨认出这是调用了qWarning()。以同样的格式，你应该在 qDebug()函数输出中插入 DEBUG 之类的字作为其第一个字。

23.1.2　qFatal()函数

最后一个 Qt 调试函数为 qFatal()。可以像使用 qWarning()和 qDebug()一样使用它。但是，qFatal()退出程序，因此，只有当发生严重错误时才应该使用它。

例如，假设程序准备用一个整数(x)除以另一个整数(y)，并且这两个整数都由程序控制。现在，如果 y 等于 0，则无法做除法，可以想象程序将出现严重问题。在这种情况下，应该像下面这样使用 qFatal()函数：

```
if ( y == 0 )
{
    qFatal( "FATAL: Can't divide x by 0!" );
}
```

这里，如果 y 等于 0，对 qFatal()函数的调用将使程序在 stdout（Windows 下为调试器）上显示 FATAL: Can't divide x by 0!信息后退出。如你所看到的，FATAL 字用于标识这是一个 qFatal()函数调用。但是，由于程序将退出，所以这将非常明显。

23.2　理解 Qt 调试宏

尽管 qglobal.h 包含很多调试宏定义，但是这一节只介绍两个重要宏：ASSERT()和 CHECK_PTR()。这两个宏（还有 qglobal.h 中的所有其他宏）使用 qDebug()、qWarning()和 qFatal()函数为某个事件创建调试信息。直接使用这些宏而不是调试函数能够节省一些工作。

23.2.1　ASSERT()宏

ASSERT()宏带一个布尔值做参数。如果值为 FALSE，它按照一定的标准显示出行号和调试信息。这里是一个例子：

```
ASSERT( x != 0 );  //This is line 48
```

这里，如果 x 是 0(注意，如果为 FALSE，ASSERT()输出语句信息)，程序文件为 test.cpp，行号为 48，则将显示以下信息：

```
ASSERT: "x == 0" in test.cpp (48)
```

23.2.2　CHECK_PTR()宏

CHECK_PTR()宏用于检查一个程序是否已经用完内存。它带一个指针做参数，如果指针为 NULL，CHECK_PTR()调用 qFatal()函数通知你程序已经用完内存，之后退出程序。来看下面的例子：

```
char *string = new char(100);
CHECK_PTR( string );
```

这里，如果 string 变量未正确创建，CHECK_PTR()将调用 qFatal()。



永远不要像下面这样在 CHECK_PTR()宏中创建对象：

```
CHECK_PTR( char *ch = new char(10) );
```

这个问题很难理解。如果你需要了解为什么不能这样使用，请参看 Qt Reference Document。要确实记住在程序中不要这样做。

23.3　用 gdb 调试器调试 Qt 程序

调试器是软件的一部分，它用于单步调试程序并查找其中可能存在的臭虫。多数免费的 UNIX 系统，包括 Linux，都带有 GNU 调试器 gdb，gdb 是 GNU 程序中的一个杰作。它不提供任何 GUI 界面，但是，在调试程序时，它是一个无价之宝。

当你准备使用 gdb 调试程序时，在编译程序阶段要打开调试符号。这由 gcc 的-g 选项设置。

23.3.1　获取和安装 gdb

gdb 很可能就在你的 UNIX/Linux 分发程序中，用系统中的标准包实用程序就能够很容易地安装它（例如，Red Hat Linux 中的 rpm 实用程序）。

但是，如果 gdb 不包含在分发程序中，则需要从 ftp://ftp.gnu.org/pub/gnu/gdb 下载。其安装过程与多数 GNU 软件一样：运行./configure，运行 make，之后再运行 make install。



在编写本书时，最新版本的 gdb 源分发程序(4.18)大小超过 11MB。因此，如果你使用低速电话拨号连接，其下载时间可能需要一个小时甚至更长。

无论你是从分发程序中安装 gdb 的二进制版本还是你自己编译源程序，都必须确保 gdb 被正确安装才能够执行这一节中的例子。

23.3.2　使用 gdb

当你要单步调试一个程序时，首先必须告诉 gdb 所执行的文件。最简单的方法是输入文件名作为 gdb 的第一个命令行参数。例如：

```
# gdb <filename>
```

因此，如果你要对 Qt 程序 MyQtProgram 使用 gdb，命令应该为：

```
# gdb MyQtProgram
```

现在，当你按回车键后，gdb 将启动，并输出以下内容：

```
GNU gdb 4.18
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i686-pc-linux-gnu"...
(gdb)
```

前面 7 行显示一些 gdb 介绍信息。最后一行，(gdb)，说明 gdb 已经启动，并准备好接收命令。注意，MyQtProgram 还没有启动。

现在，你通常需要设置断点。这意味着你指示 gdb 在指定点（这一点你知道会出现问题）停止运行。为此，你需要知道函数的内部名称。内部名称与源文件中使用的名称有一点不同，但是，用 nm 实用程序能够很容易地找到该名称。因此，启动一个新的终端仿真程序，或者切换到另一个虚拟控制台，并使用 cd 进入包含 MyQtProgram 文件的目录。为了列出 MyQtProgram 程序中的所有函数，执行下面命令：

```
# nm MyQtProgram
```

现在将有很多项在屏幕上滚动（程序越大，其中包含的项数就越多）。从一个大型列表中几乎不可能找到单个项。但是，使用 grep 命令，你能够很容易地找到所要查找的内容。假定你想查找 MyProblematicFunction()函数的内部名称，则应该输入：

```
# nm MyQtProgram | grep MyProblematicFunction
```

grep 命令能够保证只列出包含 MyProblematicFunction 字符串的行。如果列出行数多于一行，则应选择内存地址边上有 T 的一行（内存地址在列表的左边，它是一个 8 位长十六进制数）。

之后，切回到运行 gdb 的终端仿真程序或虚拟控制台，并设置断点。例如：

```
(gdb) break <name you found with nm>
```

这里，<name you found with nm>为你使用 nm 查找到的函数内部名称。如果名称正确，gdb 将响应出以下类似内容：

```
Breakpoint 1, 0x804f29c in MyQtProgram::MyProblematicFunction ()
```

现在，当断点设置之后，即可准备启动程序。为此，输入命令 run：

```
(gdb) run
Starting program: /fat/decs/qt-book/Ch23/MyQtProgram
Qt: gdb: -nograb added to command-line options.
```

```
Use the -dograb option to enforce grabbing.
```

现在，gdb 将启动程序，并向你显示详细信息说明正在发生的操作。在第 2 行，gdb 报告它将要启动程序。在接着下面的两行中，Qt 给出信息说明添加了-nograb 命令行参数（这是因为你正在通过 gdb 运行程序，详细内容请看下一节）。

当程序启动后，你应该使程序调用断点函数（这里为 MyProblematicFunction ()）。当调用时，程序将停止运行，（gdb）提示符再次显示出来。这时，可以使用各种 gdb 命令查看程序状态。不幸的是，gdb 命令是一个非常大的话题，这一节不可能完全介绍。但是，gdb 具有帮助功能，你可以使用它读取所有命令的简短信息。首先，在(gdb)提示符下输入 help，你将得到以下输出内容：

```
(gdb) help
List of class of commands:

aliases - - Aliases of other commends
breakpoints - - Making program stop at certain points
data - - Examining data
files - - Specifying and examining files
internals - - Maintenance commands
obscure - - Obscure features
running - - Running the program
stack - - Examining the stack
status - - Status inquires
support - - Support facilities
tracepoints - - Tracing of program execution without stopping the program
user-defined - - User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help" followed by command name for all documentation.
Command name abbreviations are allowed is unambiguous.
```

现在，从列表中选择一个主题，在 help 后输入这个主题。例如，如果你想更进一步了解怎样检查数据，可输入下面命令：

```
(gdb) help data
Examining data.

List of commands:

call - - call a function in the program
delete display - - Cancel some expressions to be displayed when program stops
disable display - - Disable some expressions to be displayed when program stops
disassemble - - Disassemble a specified section of memory
display - - Print value of expression EXP each time the program stops
```

```
enable display - - Enable some expressions to be displayed when program stops
inspect - - Same as "print" command
output - - Like "print" but don't put in value history and don't print newline
print - - Print value of expression EXP
printf - - Printf "printf format string"
ptype - - Print definition of type TYPE
set - - Evaluate expression EXP and assign result to variable VAR
set variable - - Evaluate expression EXP and assign result to variable VAR
undisplay - - Cancel some expressions to be displayed when program stops
whatis - - Print data type of expression EXP
x - - Examine memory: X/FMT ADDRESS

Type "help" followed by command name for all documentation.
Command name abbreviations are allowed is unambiguous.
```

这些 gdb 命令能够用来检查程序数据。例如，可以使用 call 命令执行程序中一个指定函数。使用这些命令和其他 gdb 帮助部分中的命令，你能够完全控制程序。

关于 GNU 调试器更多信息，请参看 gdb 帮助页面（man gdb）。

23.4　命令行选项

当调试一个 Qt 程序时，使用一些命令行参数能够使调试工作变得更加简单。

首先，-nograb 参数告诉程序不要响应鼠标和键盘（它使其他程序能够同时使用鼠标和键盘）。如果在运行期间响应鼠标和键盘，就不可能输入命令和点击其他程序（如调试器）。所幸的是，当通过 gdb 调试器运行 Qt 程序时，该参数被自动设置。

万一你需要程序在通过 gdb 运行时能够响应鼠标和键盘（但是，这种情况很少），-dograb 参数可使程序忽略-nograb 参数。

-sync 命令行参数告诉 X11 服务器（它只能工作在 X11 下）在同步模式下运行。这意味着 X 服务器不再使用任何缓冲优化，而是立即执行每个客户端请求。这使程序运行更慢，也因此更加容易调试。

22.5　小　结

使用 Qt 调试函数和宏能够使你，开发人员，更好地理解在不使用调试器时程序怎样运行。阅读这些函数和宏在 std(X)或调试器(Windows)上的输出信息能够提高你查找到臭虫的机会。当所运行的程序没有任何输出时，就很难了解实际所发生的事情，例如，怎样调用和什么时候调用函数。

当然，调试函数和宏的输出信息不局限于函数调用，你可以显示更详细的信息。因为调试函数使用与 printf()函数相同的参数列表，所以很容易在输出字符串中嵌入程序变量。

在这一学时，学习了 gdb 调试器的基本知识。当你感觉调试函数和宏不能满足需要，例如，你需要更内部的一些信息时，这个程序就很有帮助。掌握了 gdb，你就能够完全控制程序，战胜所有臭虫，无论它们有多么复杂。

最后，记住在程序的最后版本中删除所有调试函数和宏调用。

22.6　问题与答案

问：从哪里能够查找到所有其他在这一学时中未提到的调试宏？

答：很不幸，在编写本书时还没有关于这些宏的详细信息。但是，Qt Reference Document 的“Debugging techniques”部分简短描述了这些宏。

问：我不理解，为什么要使用调试宏代替调试函数？

答：调试宏能够为你做很多工作。使用它们，你不必自己创建输出字符串——它由宏处理。唯一需要你做的事情是向宏提供一个指针或布尔值，使它能够判断是否应该输出调试字符串。

问：我不喜欢基于文本的程序。gdb 调试器有 GUI 界面吗？

答：实际上，有。它由 Tcl/Tk 编写，可以在 http://sourceware.cygnus.com/insight/站点查找到。

22.7　作　业

完成下面的问题和练习将有助于你牢记这一学时所学的 Qt 调试技术。回答这些问题也能够使你确实理解 gdb 调试器的基本知识。

22.7.1　测验

1. 什么时候应该使用 qDebug()函数？
2. 什么时候应该使用 qWarning()函数？
3. 什么时候应该使用 qFatal()函数？
4. ASSERT()宏的作用是什么？
5. CHECK_PTR()宏的作用是什么？
6. 什么是调试器？
7. 怎样告诉 gdb 你想处理哪个文件？
8. 使用什么 gdb 命令能够设置断点？
9. gdb 中 run 命令的作用是什么？
10. 你猜想一下，哪个命令能够退出 gdb 调试器？

22.7.2　练习

1. 用一个 Qt 程序做参数启动 gdb 调试器。为程序设置断点，运行程序，要确保它能够到达断点位置。当程序停止时，试运行 help data 列出的一些 gdb 命令。例如，用 call 命令执行程序中某个函数。

第 24 学时　使用 Qt 构造程序

随着 Qt 越来越流行，对 GUI 构造程序的需求也变得更加强烈。GUI 构造程序是一个程序，它能够让你在图形环境下创建 GUI 界面，而通常不需要编写任何一行代码！但是，这些程序只能创建 GUI 界面，它们不能为你创建实际程序功能。

在这一学时，将向你介绍 3 个这样的构造程序：QtEz、AtArchitect 和 Ebuilder。这 3 个程序都能很容易地用于构造 Qt GUI 界面。选择使用哪一个只是你个人爱好问题。

注意，这一学时只介绍与 Qt 相关的 GUI 构造程序。一些程序也可用于构造 KDE，但是这些不在这里介绍（毕竟这是一本讲 Qt 的书）。

24.1　QtEz

QtEz 被很多人认为是最好的 Qt GUI 构造程序。在这一节，你将学习怎样获取、编译和安装 QtEz，以及怎样使用它创建简单的 GUI。你会感到惊讶，创建专业的 Qt 项目竟是如此简单！

24.1.1　获取和安装 QtEz

你需要做的第一件事情是获取最新的 QtEz 分发程序，它能够很容易地从 QtEz Web 站点(http://www.ibl.sk/qtez/)下载。在编写本书时，QtEz 的最新版本为 0.85.2。你能够找到 QtEz 的二进制和源分发程序。

如果你的系统能够使用二进制分发程序，就应该选择该版本。但是，如果情况不是这样，则应该下载.tar.gz 源分发程序。

如果下载了源分发程序，你需要做的第一件事是像下面这样解开文件：

```
# tar xvfz qtez-0.85-2.tar.gz
```

这将创建一个 qtez-0.85-2 目录，这里保存所有源分发程序。之后，你应该准备在系统上编译源程序。为此，你需要做一些事情。首先，运行包含在 QtEz 分发程序根目录中的 autogen.sh 脚本程序：

```
# ./autogen.sh
```

你将看到以下输出内容在屏幕上慢慢滚动：

```
Creating Makefile.in in all subdirectories...
Creating modifications for MOC files...
Creating script ./configure
```

```
Create include links...
Complete, please run ./configure or ./configure -help for help
```

像最后一行指示那样，现在你应该执行 configure 脚本程序：

```
# ./configure
```

configure 脚本程序将检查系统，并根据它所查找到的信息创建编译文件。这可能要花费较长的时间。但是，当它完成之后，你需要像下面这样执行一个叫做 automoc 的 Perl 脚本程序，它也位于 QtEz 分发程序的顶级目录中：

```
# perl automoc
```

这能够确保 MOC 文件被正确处理。最后，你即可开始编译操作。与通常一样，这只需运行 make：

```
# make
```

现在，编译已经启动。根据你系统的速度和负载情况不同，这可能需要一些时间。在编译完成后，即可开始在系统上安装二进制文件。这用下面命令来实现：

```
# make install
```

所有需要的文件都将被拷贝到系统上的合适位置。在安装完成后，你需要设置 QtEz 环境变量。它应该被设置到 QtEz 模板文件所在的目录。这些文件未被自动安装，因此，你需要手工安装这些文件：

```
# cd qtez-0.85-2/templates
# mkdir /usr/local/share/qtez
# cp -r * /usr/local/share/qtez
```

之后，将 QTEZ 设置到你刚创建的目录。如果使用 bash，则用下面命令：

```
# export QTEZ=/usr/local/share/qtez
```

如果你使用 C 外壳，则用以下正确命令：

```
# setenv QTEZ=/usr/local/share/qtez
```

如果你需要多次使用 QtEz，很可能需要将这一行代码添加到一个设置 QTEZ 变量的启动文件中。

24.1.2　用 QtEz 创建简单的 GUI

在这一节，将学习 QtEz 基本用法。将一步一步地指导你怎样创建简单的 GUI。这些信息能够使你逐渐熟悉 QtEz。

首先，你需要启动 QtEz：

```
# QtEz
```

现在 QtEz 应该已经启动，它将在屏幕上显示两个很大的窗口。这些窗口如图 24-1 所示。

上面的窗口是 QtEz 主窗口，从中控制整个程序。下面的窗口用于设置环境选项。如你在 QtEz 主窗口（上面的一个窗口）中所看到的，这是一个具有很多选项和功能的大程序。但是，尽管具有这些功能，QtEz 仍是非常直观和容易使用的。

一、创建新项目

为了创建新项目，点击 QtEz 主窗口中的 File 菜单，然后选择 New，Project。现在，打开一个新对话框。这个对话框如图 24-2 所示。

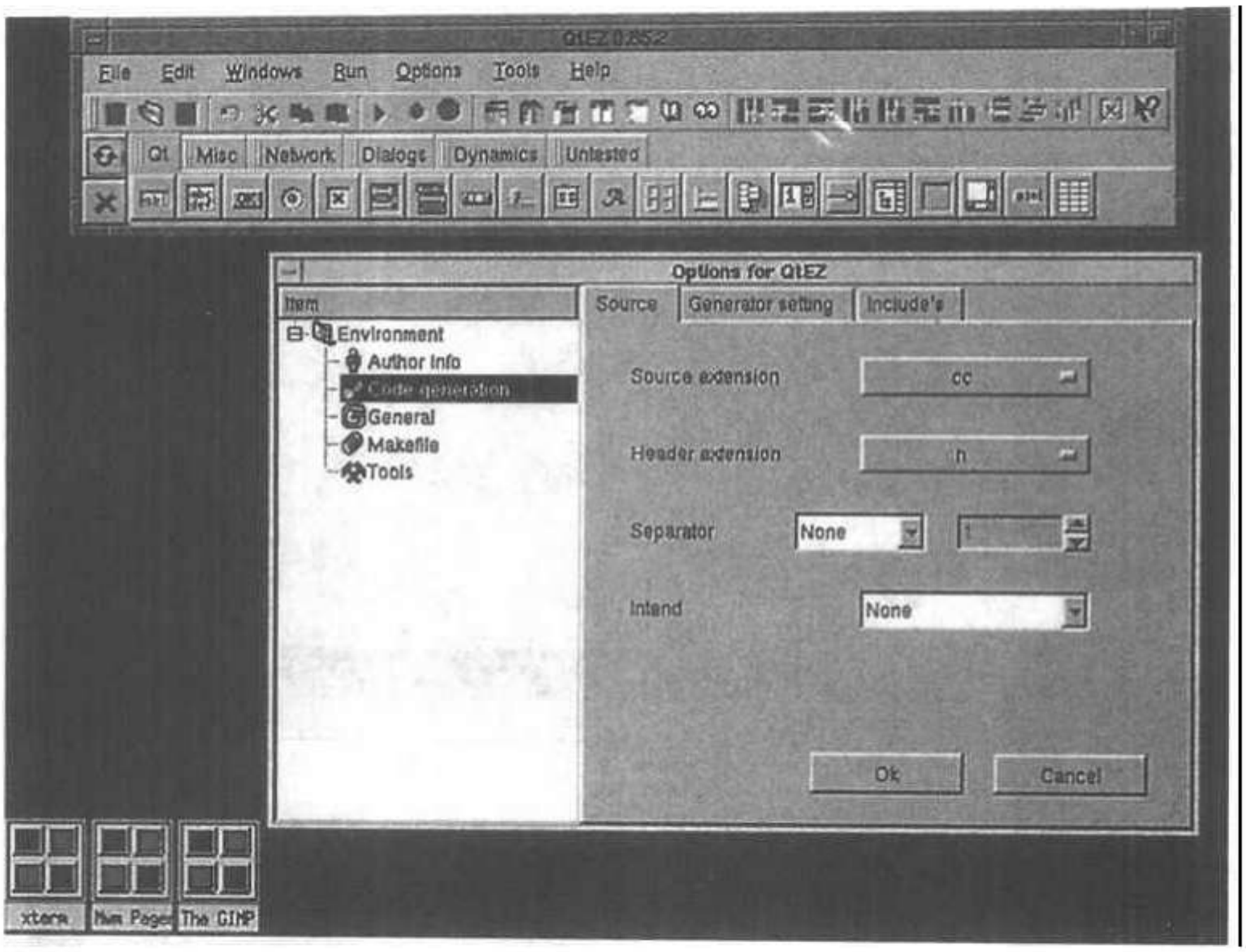


图 24-1　两个标准 QtEz 窗口

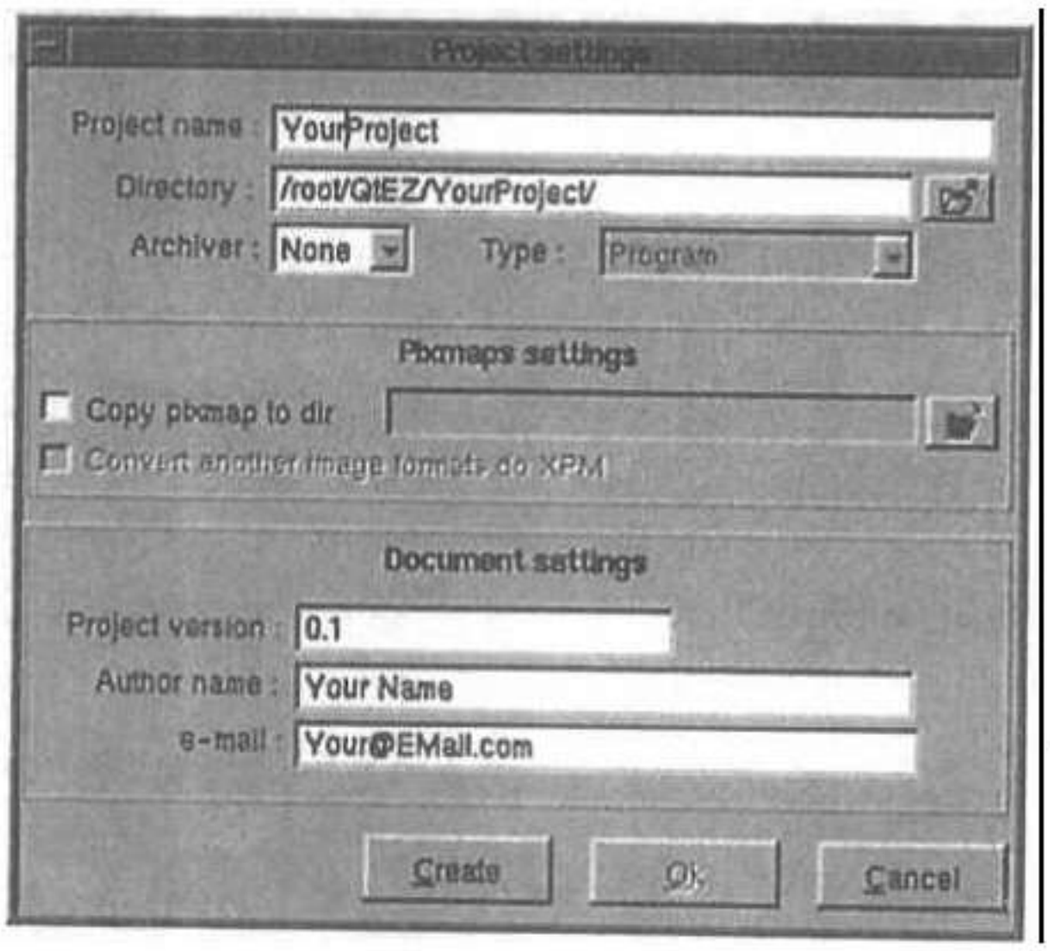


图 24-2　QtEz 的 Project Settings 对话框，用于设置新项目的通用信息

这是 Project Settings 对话框。你在这里设置项目的通用信息，如项目名称、存储位置、创建者名称和电子邮件地址。输入这些信息后点击 Create 按钮。

现在，弹出两个新窗口，如图 24-3 所示。

当这两个窗口出现时，QtEz 将立即检查系统，并创建编译文件和其他标准文件和目录。该操作的输出信息显示在其中一个新窗口上。另一个窗口代表（实际上是）你的项目。如你

所看到的，这个窗口仍然是空的。

为了了解你选择创建新项目时 QtEz 所做的工作，来浏览一下在 Project Settings 窗口中你为项目所选择的目录。在该目录下运行 ls 命令，它将显示以下信息：

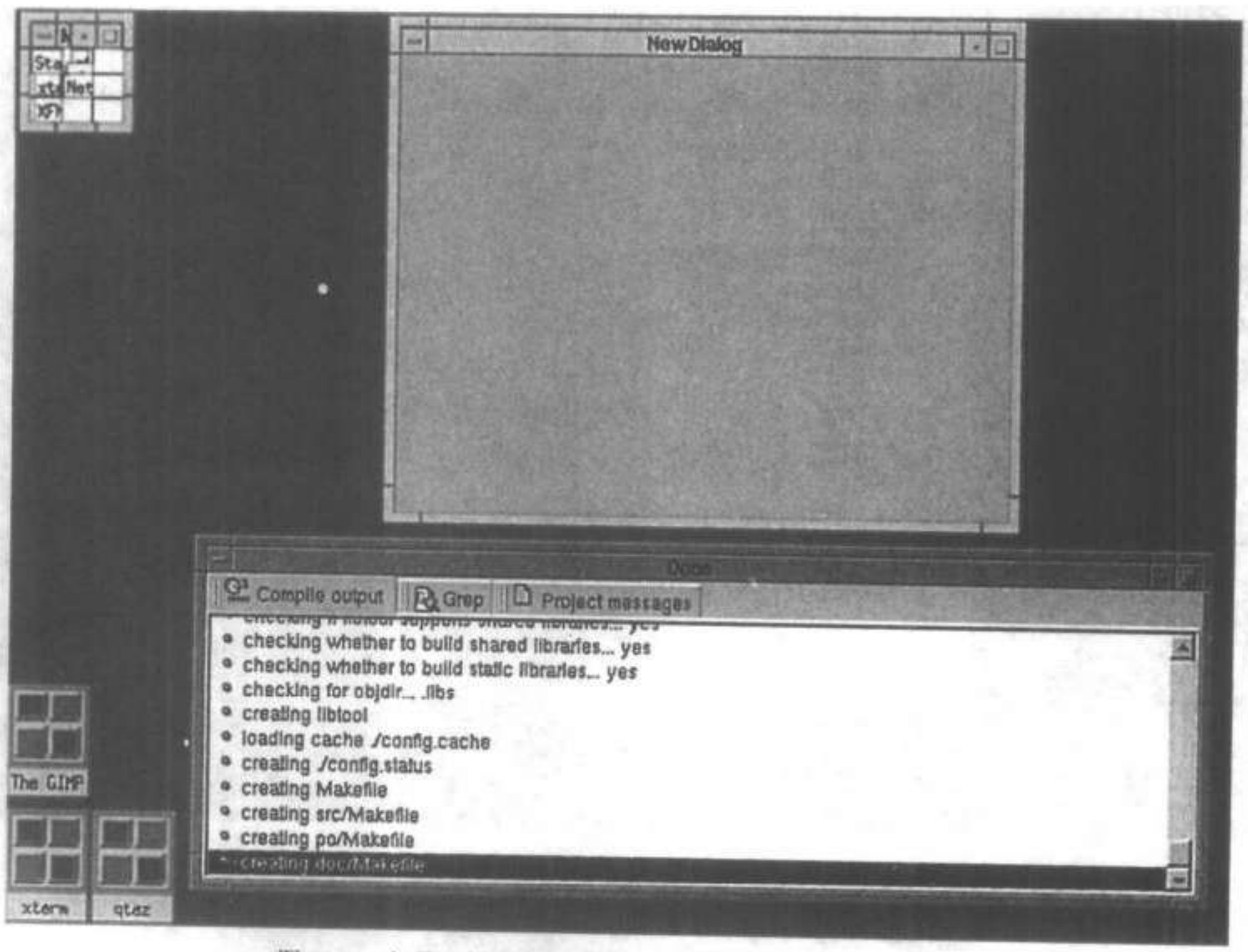


图 24-3　空项目窗口（上面窗口）和项目编译输出信息窗口

```
# ls
total 790
 1  AUTHORS          1  README              3  config.cache    6  install-sh*
19  COPYING          1  YourProject.lsm    30  config.guess*  114 libtool*
 1  ChangeLog        1  YourProject.qtz     5  config.log     90  ltconfig*
 8  INSTALL         93  acinclude.m4        7  config.status* 106 ltmain.sh
14  Makefile        95  aclocal.m4         21  config.sub*     7  missing*
 1  Makefile.am     38  am_edit*           96  configure*      1  mkinstalldirs*
14  Makefile.in      1  autogen.sh*         1  configure.in    1  po/
 1  NEWS            11  automoc*            1  doc/            1  src/
```

实际上，这是一个完整的 Qt 项目，它具有编译文件、configure 脚本程序和放置源程序、文档以及翻译文件的目录。但是，这个项目还没有任何实际功能。这还需要你来实现！

二、向项目添加部件

一旦一个新项目创建之后，你就能够向它添加部件。这分两步进行：先从 QtEz 主窗口中选择部件，之后在空窗口（项目）中绘制它。

首先从 QtEz 主窗口中选择部件。这些部件显示在窗口底端，并在选项栏上将它们分为

不同的种类。选项栏标签分别为 Qt、Msic、Network、Dialogs、Dynamics 和 Untested。现在，我们使用 Qt 选项栏中的部件。从 Qt 选项栏所包含的部件中选择按钮图标（如果你不能确定是哪个图标，可将鼠标移到图标上面，很快就会显示出该图标的描述文本）。

现在，将鼠标移到空的项目窗口上，按下鼠标左键并拖动鼠标，然后松开鼠标。这样就能够在窗口上创建一个按钮，就是这样简单！拖动按钮的一角就能够调整按钮大小，或者将按钮拖到窗口中的某个位置来移动按钮。一个按钮例子如图 24-4 所示。

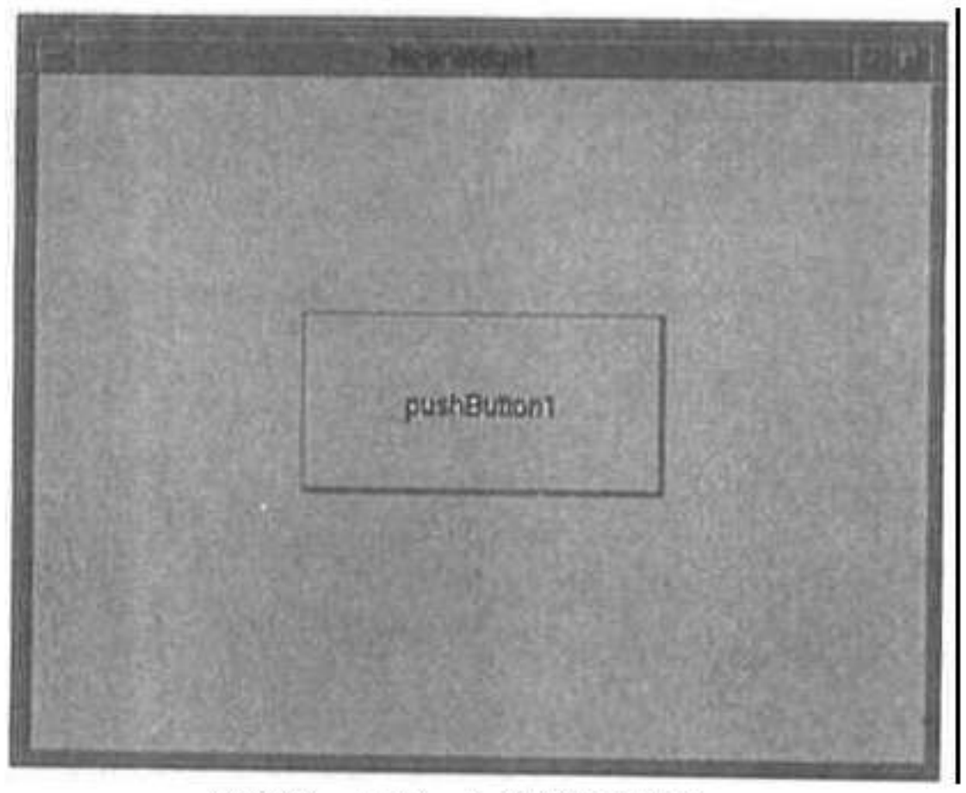


图 24-4　具有一个按钮的项目窗口

为了改变按钮设置，如按钮标签等，用鼠标右键点击按钮，从显示出的菜单中选择 Attributes。现在将打开按钮的 Attributes 对话框（所有部件都具有 Attributes 对话框，在该对话框中可以改变部件属性）。该对话框如图 24-5 所示。



图 24-5　按钮的 Attributes 对话框

在这个对话框中，将对话框向下滚动到 Text 行，单击该行右边的表元。这时，该表元变为可编辑状态，在这里可以输入按钮标签。输入标签后按回车键即可激活所做的修改，之后关闭该对话框。现在，按钮标签就被改变。

这里所采用的添加和定制按钮的过程与所有其他部件相同。因此，如果你理解这一节中所介绍的内容，添加其他部件也毫无问题。

三、连接信号和槽

使用 QtEz 所带的 Signal/Slot 编辑器，将使信号和槽的连接操作变得非常简单。从 QtEz 主窗口中启动 Signal/Slot 编辑器，只需点击工具栏（位于菜单栏下面）中从左边数第 14 个图标。Signal/Slot 编辑器窗口如图 24-6 所示。

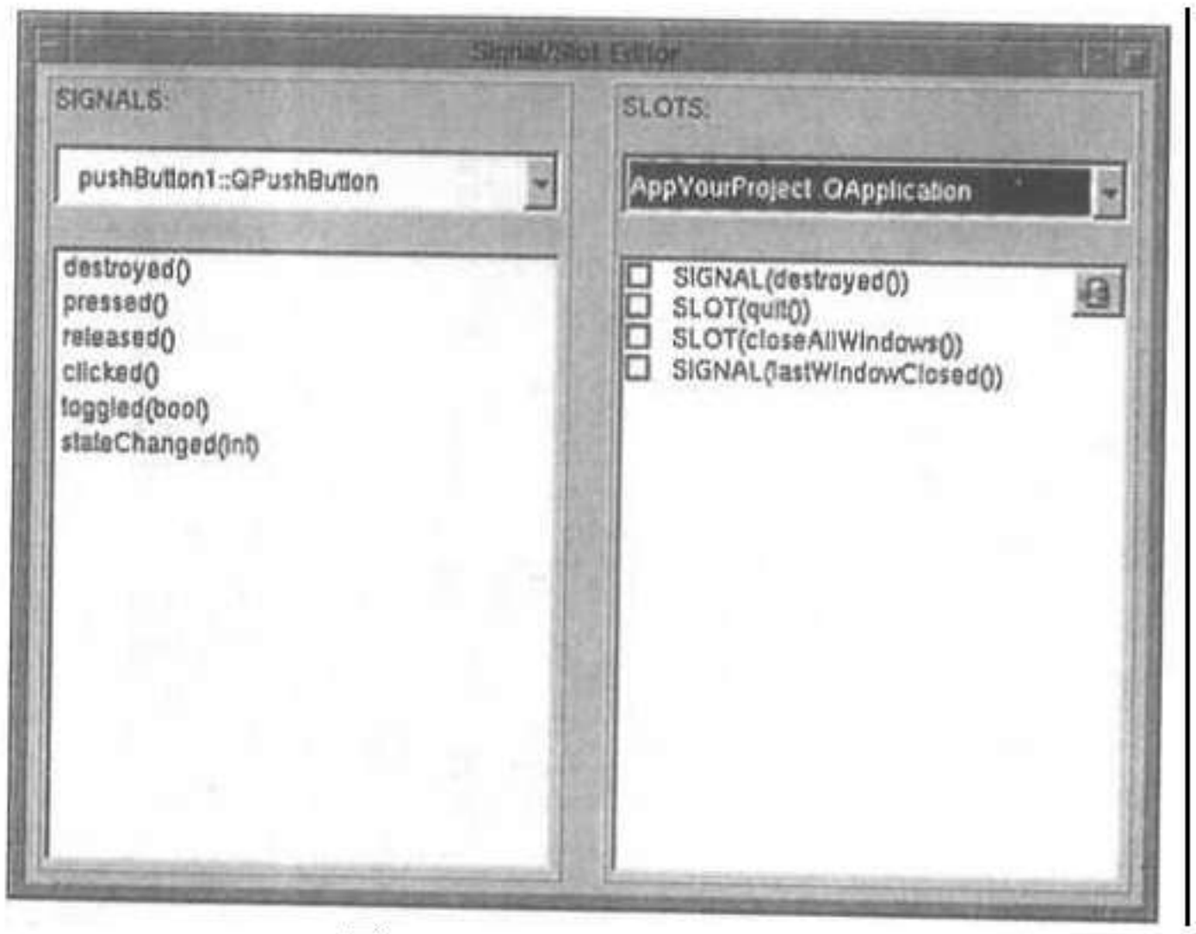


图 24-6　Signal/Slot 编辑器窗口

在窗口的上部有两个下拉菜单，用它们可以选择所要处理的部件。点击左边的下拉菜单并选择按钮（名称可能为 pushButton）。该按钮信号将显示在列表框中。从这个列表框中选择 clicked()信号。现在，选择 QApplication 对象（名称为 AppYourProject，如果项目名称为 YourProject 的话），之后双击 quit()槽（以使槽名左边框中显示出一个绿色的叉号）。

现在已经选中左边的 clicked()信号和右边的 quit()槽。在此之后，关闭 Signal/Slot 编辑器窗口。这样，按钮的 clicked()信号将被连接到 QApplication 对象的 quit()槽。尽管在前面已经这样做了很多次，但这是第一次在图形界面下实现它。

四、创建 QtEz 项目源文件

为了编译新的 QtEz 项目，首先需要创建其源文件。这通过 Run，Create，Source Files 菜单实现。选择该菜单项，QtEz 将创建项目源文件，并将它们放置在项目根目录（在 Project Setting 窗口中所选择的目录）中的 src 子目录下。因此，如果项目根目录为 /home/usr/QtEz/YourProject，则源文件目录将创建在/home/usr/QtEz/YourProject/src 下。如果你按照这里的指导创建一个只有单个按钮的简单 QtEz 项目，该目录中的内容将为：

```
# ls
total 34
    14  Makefile     14  Makefile.in 1   main.h   1   widget1.h
    1   Makefile.am  1   main.cc 1   widget1.cc  1   widget1_data.cc
```

为了查看这些文件中的实际 Qt 代码，我们打开 main.cc 文件，其中具有 main()函数。该文件内容如程序清单 24-1 所示：

程序清单 24-1　　QtEz 所产生的 main() 函数 QtEz

```
 1: /*****************************************************
 2: ** Source Dump From QtEZ http://www.ibl.sk/qtez **
 3: **---------------------------------------------------**
 4: **  Dumped: Wed Feb 13 14:18:20 2000
 5: **      To: main.cc
 6: ** Project: YourProject
 7: ** Version: 0.1
 8: **  Author: Your Name
 9: **  e-mail: Your@Name.com
10: *****************************************************/
11:
12: /*** Main Include ****/
13: #include "main.h"
14:
15: /*** Top Level Widget Includes ***/
16: #include "widget1.h"
17:
18: int
19: main(int argc,char **argv)
20: {
21:  QApplication AppYourProject(argc,argv);
22:  AppYourProject.setStyle(new QWindowsStyle);
23:  Cwidget1 widget1(0, "widget1");
24:
25:  QObject::connect(widget1.pushButton1,SIGNAL(clicked()),
26:         &AppYourProject,SLOT(quit()));
27:  AppYourProject.setMainWidget(&widget1);
28:  widget1.show();
29:  int retCode = AppYourProject.exec();
30:  return(retCode);
31: }
```

QtEz 所产生的所有文件中的前面几行都是文件的描述信息（第 2 行到第 9 行），如文件创建时间（第 4 行）、它属于哪个项目（第 6 行）、作者是谁（第 8 行）等等。如你所看到的，main.cc 文件也是这样。这些行只是普通的 C++/Qt 代码，尽管它们与本书所使用的风格不同，但它们是有效的。

如果需要，你也可以查看其他源文件，以了解 QtEz 所构造的项目。尽管在使用 GUI 构造程序时不必控制源代码，但是，对这些文件有一个总的了解有助于你修改可能存在的 QtEz 不能自动修改的问题。

五、编译和运行 QtEz 项目

可以采用两种方法编译和运行 QtEz 项目：直接在 QtEz 下编译和运行，或者在命令提示符下手工编译和运行。

为了在命令提示符下编译和运行 QtEz 项目，需要遵守多数 UNIX/Linux 源分发程序所使用的标准——也就是运行 configure、make，之后再运行 make install。当然，应该在项目的根目录下运行这些命令。但是，对于这个测试项目你很可能不想执行 make install，而是直接从 src 目录中运行它。因此，为了手工编译和运行项目，使用 cd 切换到项目的根目录，并执行下面命令：

```
# ./configure
# make
# cd src
# ./YourProject
```

当然，需要将 YourProject 修改为真正的项目名称。

从 QtEz 中编译和运行项目更简单。只需要点击一个按钮。因此，从工具栏中找到这个绿色按钮（从左边数第 8 个），并点击它。你可能想显示出 Compile 窗口，以便看到所发生的操作。如果 Compile 窗口未显示出来，从 Windows 菜单中点击 Compile Window 菜单项，Compile 窗口就将显示出来，它显示编译器在编译期间所输出的编译信息。当编译完成后，程序被自动执行显示到屏幕上（如图 24-7 所示）。

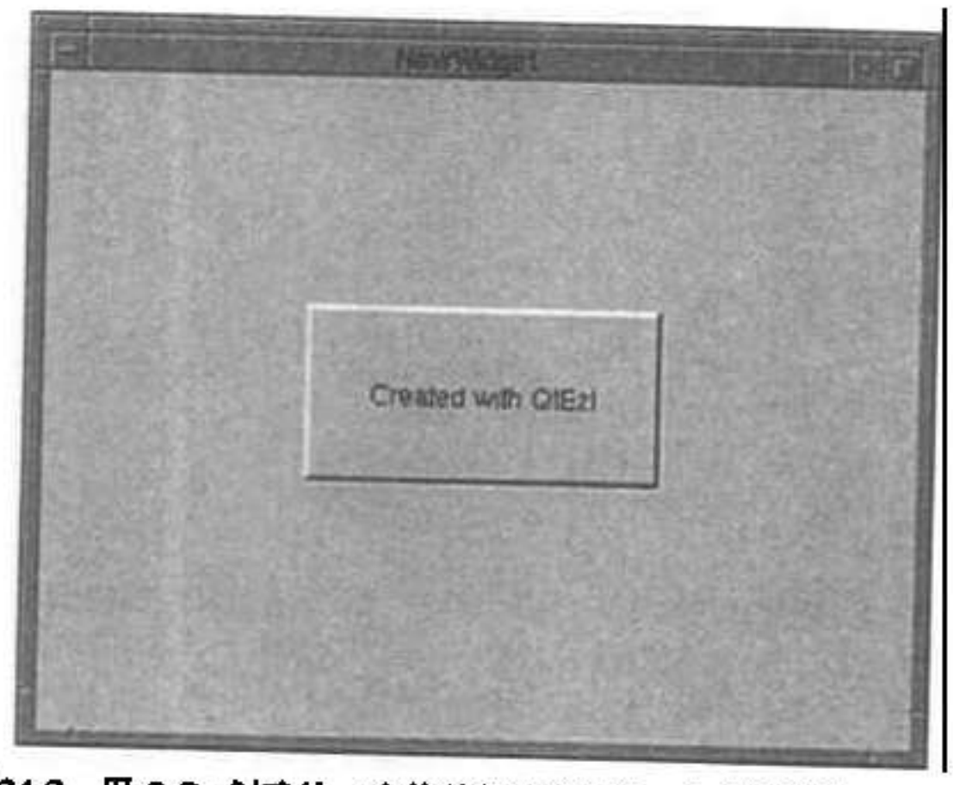


图 24-7　用 QtEz 创建的一个简单的 Qt 程序，点击按钮将退出程序

这就是用 QtEz 创建的第一个 Qt 程序！它由一个 QWidget 对象、一个 QPushButton 对象和一个 QApplication 对象组成。QPushButton 对象的 clicked()信号被连接到 QApplication 对象的 quit()槽。你对这个概念并不陌生，但这次使用 QtEz 创建。

24.2　QtArchitect

另一个流行的 Qt GUI 构造工具是 QtArchitect。尽管它不像 QtEz 那么大，但它是一个很好的软件。与 QtEz 界面相比，你可能更喜欢使用 QtArchitect 所提供的界面。但要注意，QtArchitect 不是一个完整的 Qt 项目构造程序，它只能用于构造对话框。因此，你还需要自己编写一些代码（如编写 main()函数）。

24.2.1　获取和安装 QtArchitect

http://www.qtarch.intranova.net/站点提供 QtArchitect 的二进制版本和源分发程序包。如果二进制版本适用你所用的系统，则可选择二进制版本。但是，如果情况不是这样，编译源程序也没有任何问题，只要遵照这一节中的指导即可。

首先，下载最新的源分发程序，在编写本书时为 2.0-1。文件大概只有 700KB 左右，因此，即使你使用低速连接，下载时间也不会超过几分钟。

当下载完成之后，解开文件：

```
# tar xvfz qtarch-2.0-1.tar.gz
```

这将创建一个 qtarch-2.0 目录，并将所有源文件放置在该目录中。使用 cd 命令切换到这个目录。现在，如果 QTDIR 变量已经正确设置，QtArchitect 即可用于编译。但是，最好检查一下 Makefile 文件，以确定它是否需要做进一步的修改。之后，开始编译：

```
# make
```

编译需要一段时间。当它完成后，在 qtarch-2.0 目录下有一个叫做 qtarch 的二进制文件，这就是实际程序。如果需要，也可以运行 make install 安装二进制文件（在/usr/local/bin 下）和其他几个文件（在//usr/local/lib/qtarch 下）。但是，也完全可以直接运行二进制文件。

24.2.2　用 QtArchitect 创建简单的 GUI

现在，你即可启动 QtArchitect。如果将它安装在/usr/local/bin 目录下，你可以从文件系统中的任意位置启动它。如果不是这样安装，则必须将二进制文件的位置显式告诉外壳。因此，用 cd 切换到 qtarch-2.0 目录，并执行下面命令：

```
# ./qtarch
```

QtArchitect 主窗口将显示在屏幕上。这个窗口如图 24-8 所示。

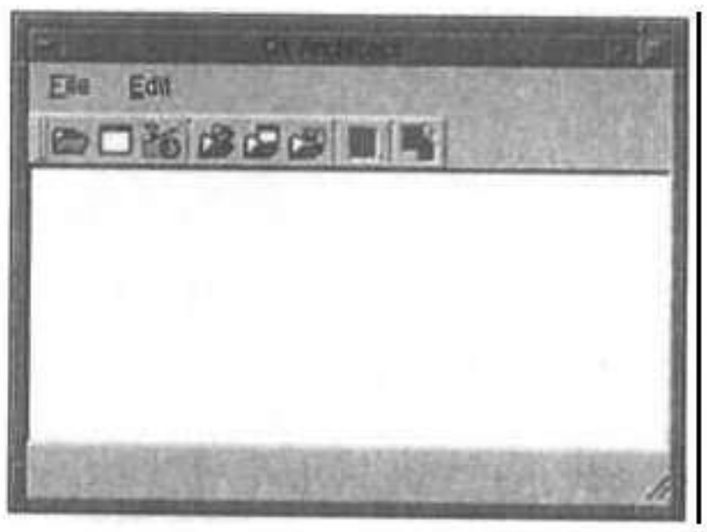


图 24-8　QtArchitect 主窗口

现在，需要做的第一件事是创建一个新项目。这通过选择 File 菜单中的 New Project...菜单项来实现。一个小图标将显示在窗口的空白区域。该图标代表你所创建的新项目。

之后，需要向项目中添加对话框。为此，从 File 菜单中选择 New Dialog...菜单项。这样做之后，将显示出对话框创建窗口，如图 24-9 所示。

如你所看到的，对话框仍然为空的。但是，向它添加部件非常简单。为了添加按钮，选择 Insert 菜单，之后选择 Button，PushButton。一个新的按钮将显示在左上角。拖动按钮将它放置在窗口的中间位置，之后拖动它的四角将其大小改变到你满意为止。下一步，右击按钮并选择 Properties。将显示出按钮的 Properties 对话框（如图 24-10 所示）。

现在，选择 Properties 对话框中标签为 Push Button 的页面，如图 24-10 所示。在对话框上端的编辑行中输入按钮的文本标签，之后点击 OK 按钮。再回到对话框创建窗口，这次将看到按钮上添加了一个标签（如图 24-11 所示）。

右击按钮，再次选择 Properties。这一次选择 Mapping 页面。在这里将按钮的信号连接到槽。点击 Add 按钮显示出 Singal/Slot Connection 对话框。从对话框的左边选择按钮的 clicked()信号，之后在下端的文本编辑框中输入信号所连接的槽名称。Singal/Slot Connection 对话框如图 24-12 所示。

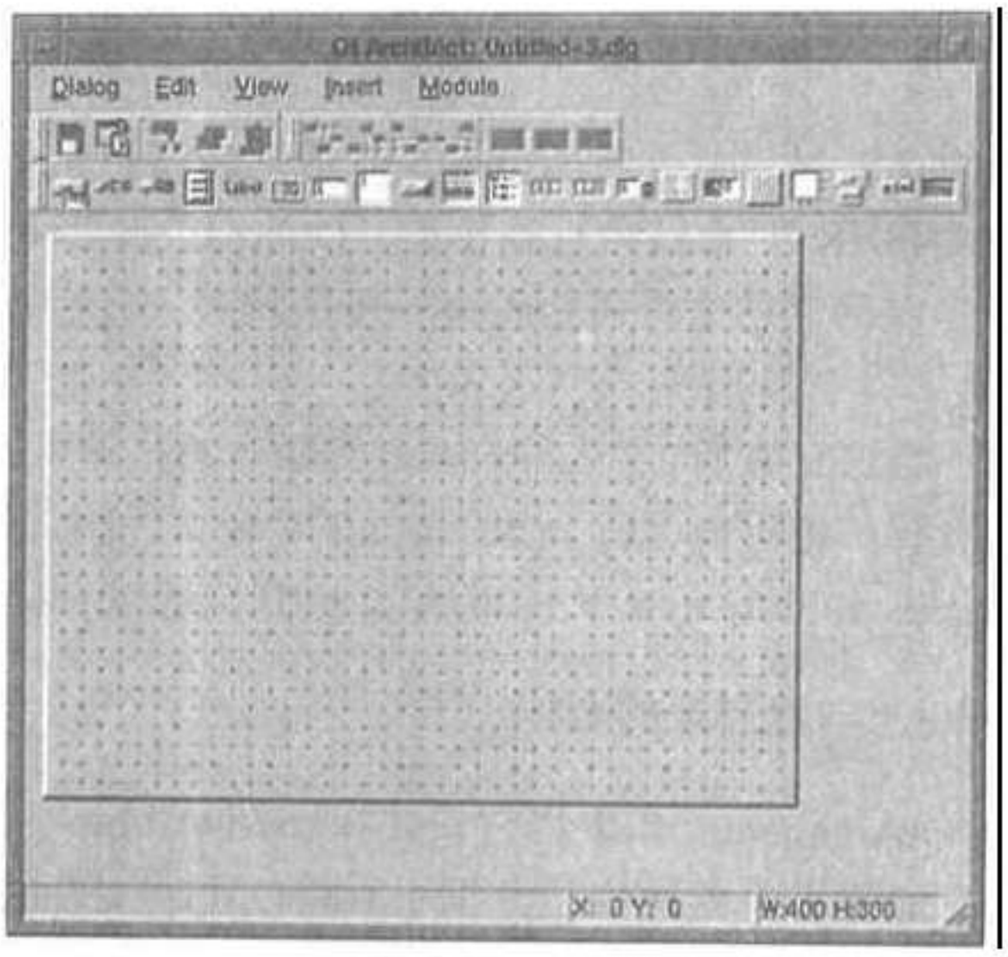


图 24-9　QtArchitect 的对话框创建窗口

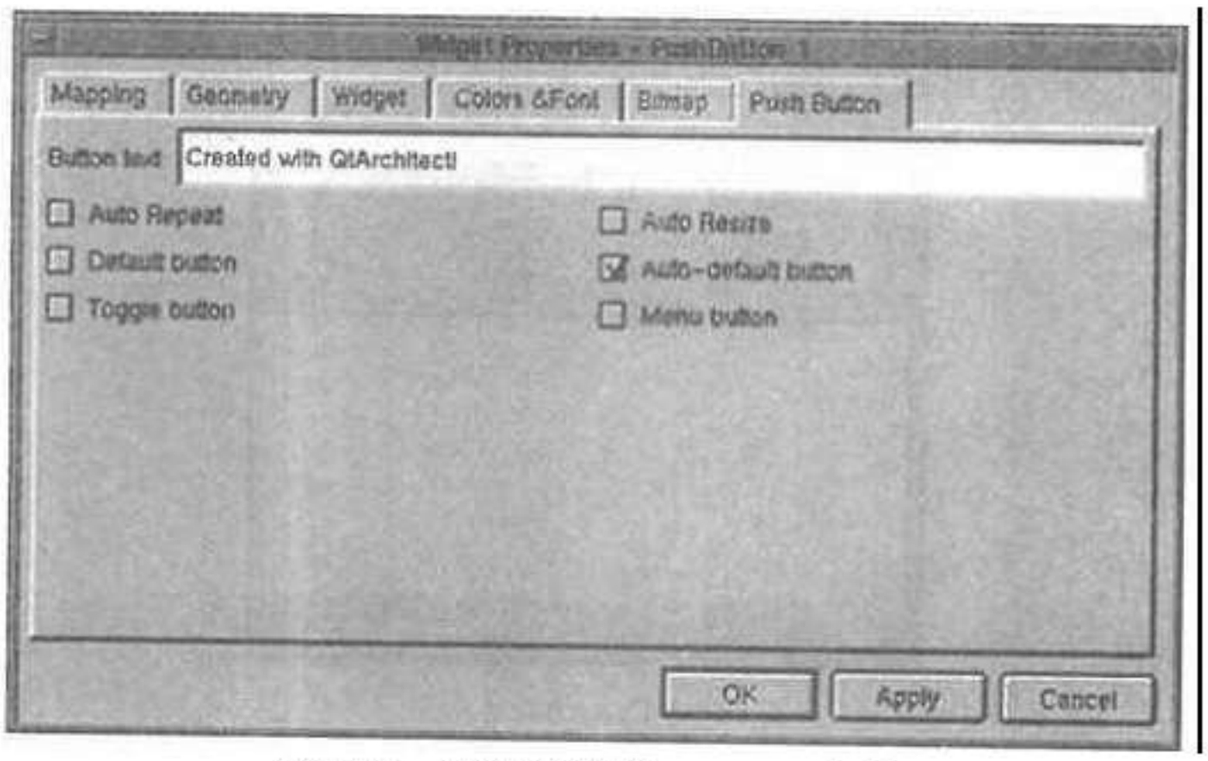


图 24-10　新创建按钮的 Properties 对话框

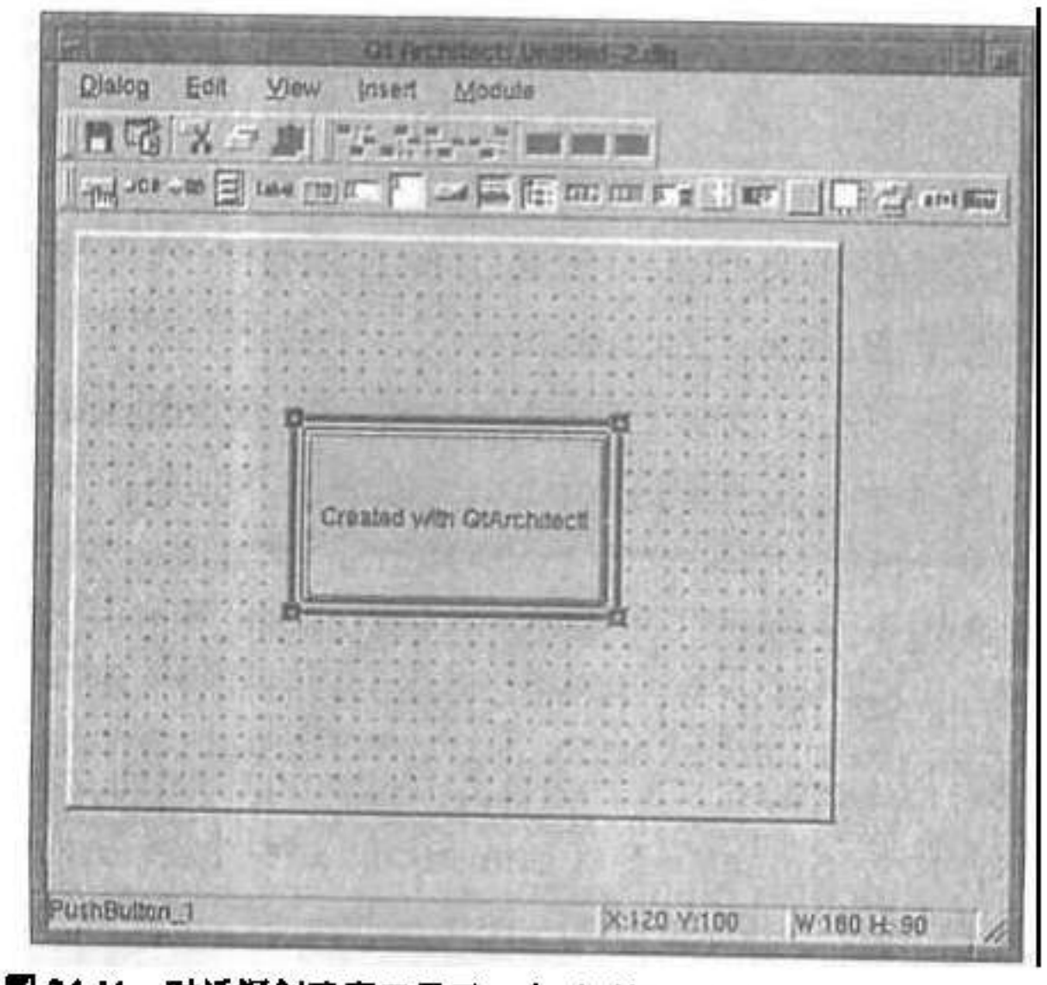


图 24-11　对话框创建窗口显示一个对话框，它有一个有标签的按钮

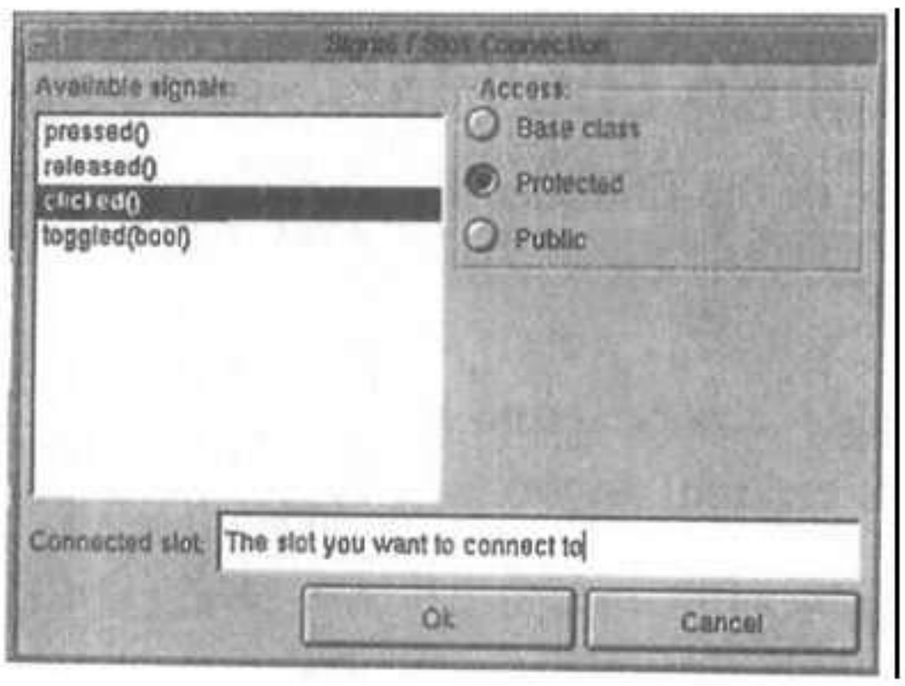


图 24-12　Singal/Slot Connection 对话框。按钮的 clicked()信号。当前被连接到 QApplication 对象的 quit()槽

下一步，点击 Singal/Slot Connection 对话框的 OK 按钮，之后再点击 Properties 对话框中的 OK 按钮，回到对话框创建窗口。在这里，从 Dialog 菜单中选择 Generate File 选项，将显示出 Generate Source 对话框。在这个对话框中你可以指定 QtArchitect 所产生的源文件名称。如图 24-13 所示。

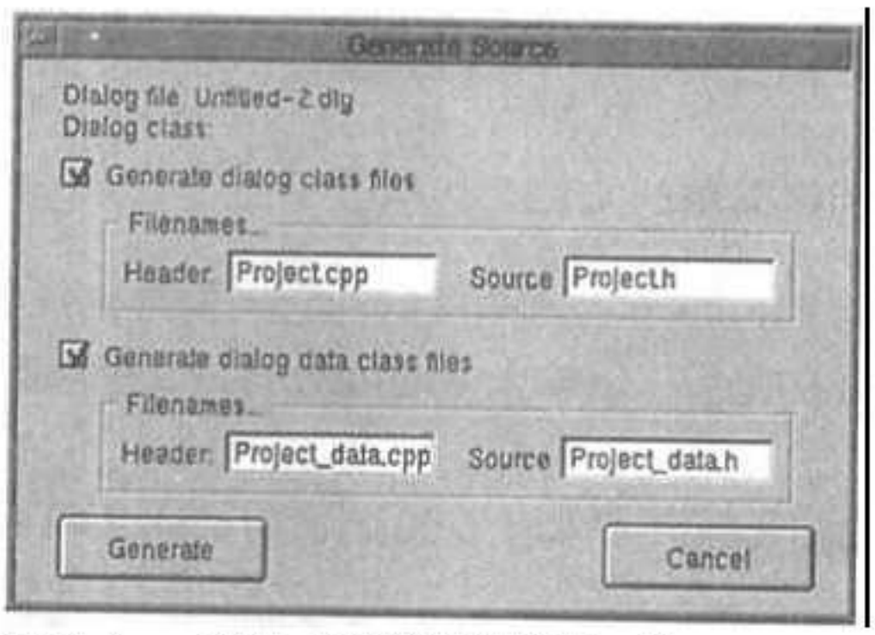


图 24-13　Generate Source 窗口，你应当像这里所显示的那样输入适当的文件名称

当完成之后，点击 OK 按钮，QtArchitect 将在你执行 qtarch 的目录中产生源文件。这样，新创建的部件将被分为 4 个文件。为了编译和使用这个部件，你需要手工创建 main() 函数。

24.3　EBuilder

在这一学时中你将学习使用的最后一个构造程序是 Ebuilder。尽管其源分发程序很小（大约为 100KB），但它非常有用，并且界面友好。这一节，将学习使用这个小巧玲珑的程序。

24.3.1　获取和安装 EBuilder

与通常一样，你需要 EBuilder 分发程序。如果二进制文件能够适用于你的系统，那就更好。但是，如果情况不是这样，则需要自己编译源文件，这一节将介绍这一内容。源和二进制分发程序都可以从 www.fys.ruu.fl/-meer/Ebuilder/中找到。要确保你得到的是最新版本！在编写本书时，其最新版本为 0.56c。

当文件下载之后，像通常一样需要解压：

```
# tar xvfz ebuilder-0.56c.tar.gz
```

这将创建一个 ebuilder-0.56c 目录，并将所有分发程序放置在该目录中。使用 cd 命令进入这个目录，并用你所喜爱的编辑器（如 vi）来编辑 config.mk 文件：

```
# vi config.mk
```

如果 QTDIR 变量已经设置，所需做的唯一事情就是修改编译器名称和 flex 实用程序路径。缺省时，编译器被设置为 cc。但是，你很可能需要将它修改为 gcc、g++等之类的编译器。flex 实用程序的路径缺省时被设置为/usr/local/bin/flex。但你很可能需要将它修改为/usr/bin/flex。如果你不能确定所用编译器名称和 flex 路径，可用 find 实用程序在系统中搜索 gcc 和 flex（更多信息请参看 man find）。

当 config.mk 文件编辑之后，即可运行 make 开始编译：

```
# make
```

现在，将开始编译，这一过程需要一定的时间。

当编译结束后，在 ebuilder-0.56c 目录中的 bin 目录下创建有两个二进制文件：ebuilder 和 convert。唯一需要关心的是 ebuilder，这是实际程序（convert 实用程序用于将旧版本的 EBuilder 项目文件转换为新格式）。为了安装 EBuilder，只需将 ebuilder 拷贝到适当目录（如/usr/local/bin）即可。

24.3.2　用 EBuilder 创建简单的 GUI

一旦在系统上正确安装 EBuilder 后，即可启动它。如果已经将 ebuilder 拷贝到 PATH 变量所指定的目录中，则只需在提示符下输入 ebuilder，并按回车键：

```
# ebuilder
```

现在，EBuilder 主窗口将显示在屏幕上。这个窗口如图 24-14 所示。



图 24-14　EBuilder 主窗口

要创建一个新部件，点击 File 菜单中的 New...之后将打开类创建窗口。在这里，告诉 EBuilder 所创建的新类名称，以及它基于哪个类。你还可以输入该类的文档信息（见图 24-15）。

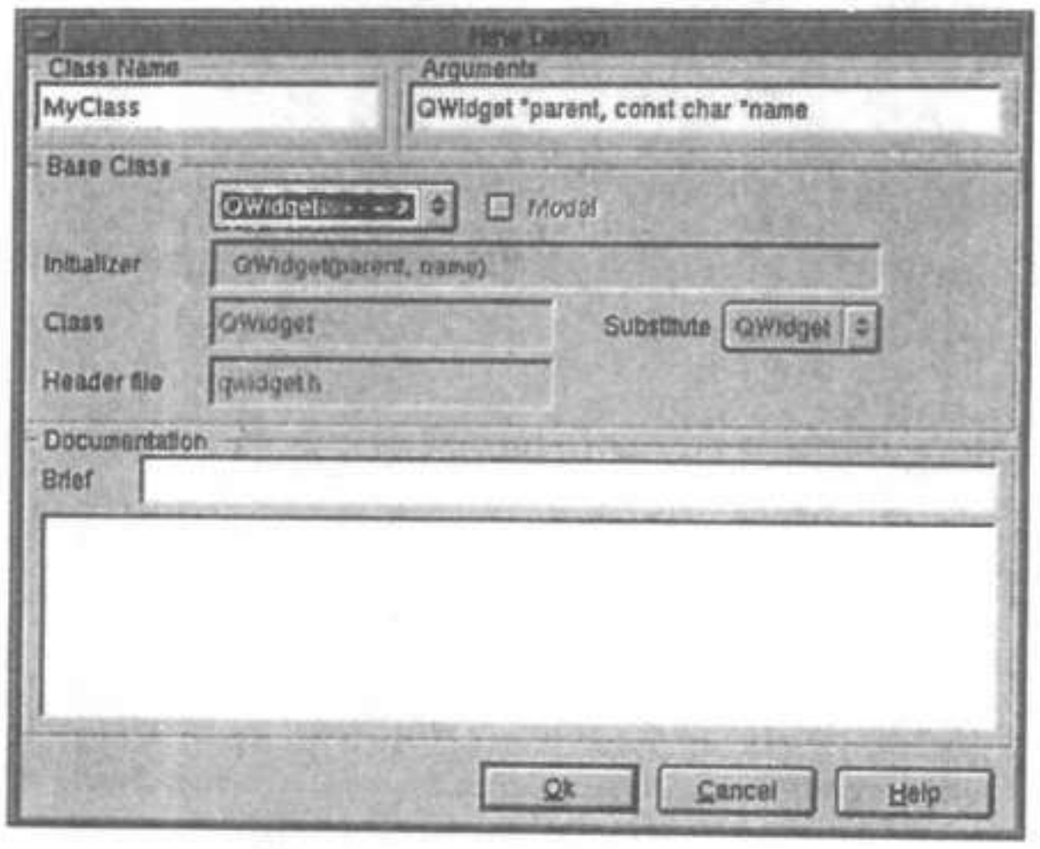


图 24-15　EBuilder 的类创建窗口

当为新类选择名称和基类后，点击 OK 按钮。这时将显示出一个空窗口。这就是你能够添加部件的新窗口。在 EBuilder 中，点击 EBuilder 主窗口中的 Edit 菜单，然后选择 Add 来添加部件。这样做之后，将显示一个部件列表。从这个列表中选择 PushButton。现在，EBuilder 将询问你新按钮的名称。如图 24-16 所示。

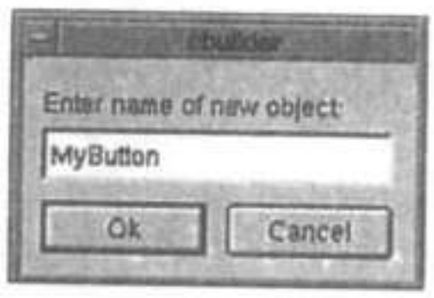


图 24-16　按钮名称输入窗口，这里所输入的按钮名称为 MyButton

输入按钮名称之后，点击 OK 按钮。现在，将显示出新按钮的 Properties 对话框。这个对话框被分为几个页面。点击 Button 页面，并在该窗口上端的文本输入框中输入按钮标签（如图 24-17 所示）。

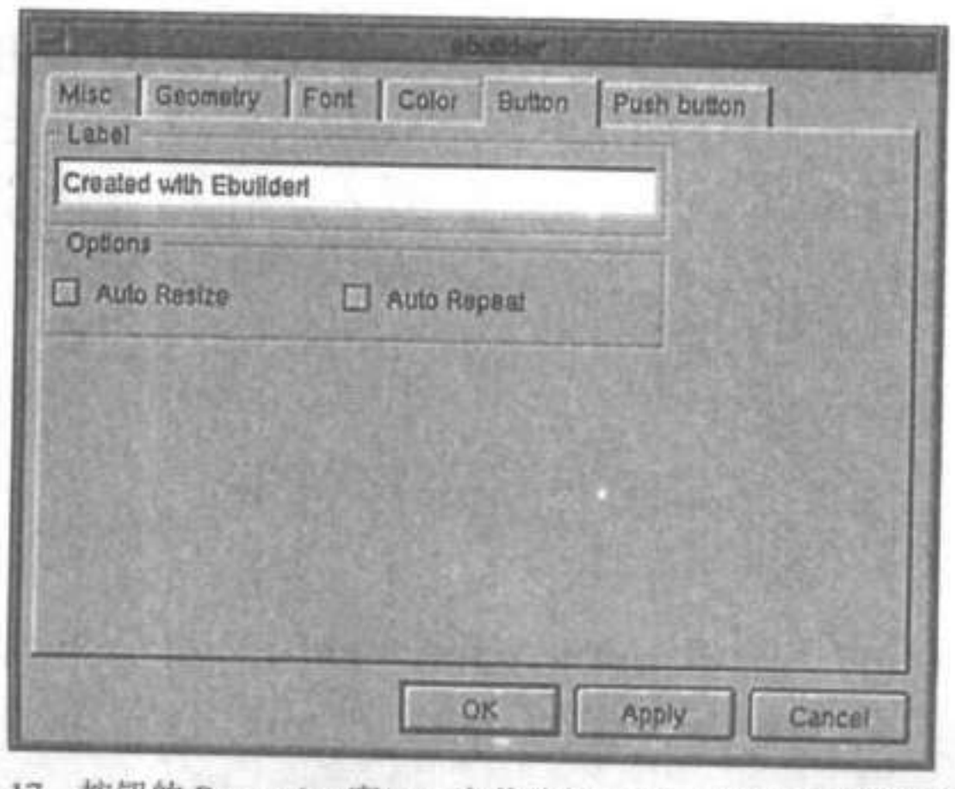


图 24-17　按钮的 Properties 窗口，当前选择 Button 页面并已经输入标签

输入按钮标签之后，点击 OK 按钮。现在，按钮将显示在项目窗口的左上角。点击按钮，使它获得焦点，然后将它拖到你想要放置的位置。拖动按钮的一角还可以调整按钮大小。当这些都做完之后，窗口看起来将如图 24-18 所示。

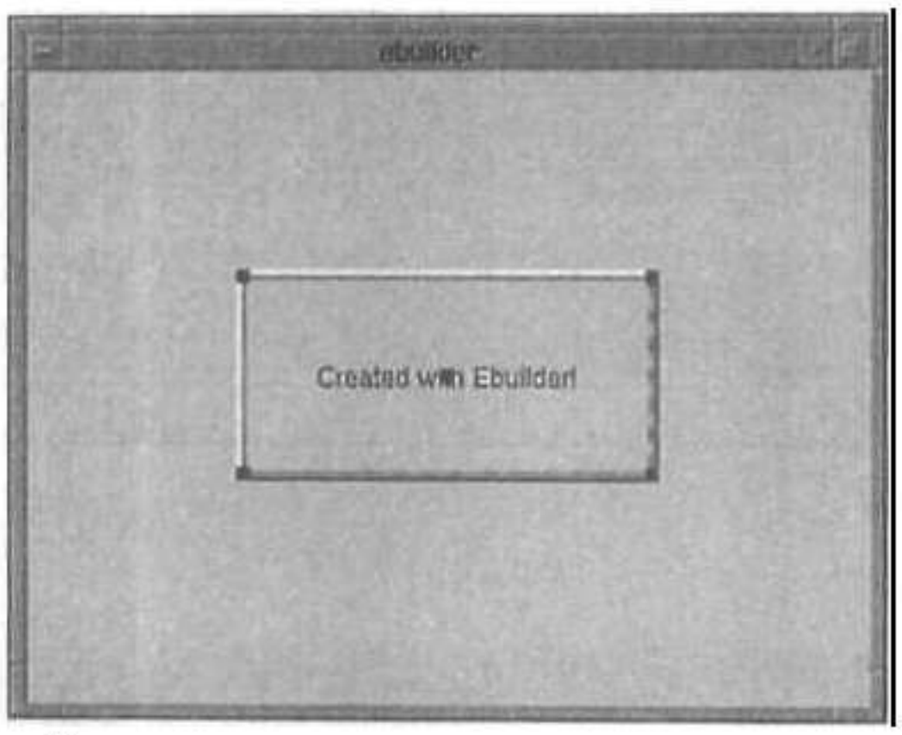


图 24-18　新创建的窗口，它具有一个有标签的按钮

现在，将这个窗口导出到一个源文件中，点击 File 菜单并选择 Export。EBuilder 将询问你所使用的源文件和头文件名称。例如，可以将它指定为 MyClass。当输入文件名称后，点击 OK 按钮。现在，将在你执行 EBuilder 的目录下创建一个源文件和一个头文件。你应该查看一下这两个文件，以了解 EBuilder 是怎样组织项目。这个窗口即可用在你的项目中。

24.4　小　结

当创建 GUI 界面时，这一学时所介绍的 GUI 构造程序能够（并将）给你很大的帮助。尽管仍需你编写一些代码才能赋予程序一些实际功能（GUI 构造程序不能编写 WWW 浏览器中的 HTML 分析器），但 GUI 构造程序能够处理 GUI 界面。这包括布置部件、设置部件大小和创建对话框等。如果你在程序中使用复杂布局和大量对话框，手工实现将花费大量时间。因此，GUI 构造程序能够为你节省大量工作量。

尽管 GUI 构造程序有很大帮助，但是，如果不了解库工作方式仍将无法创建 Qt 程序。因为 GUI 构造持程序中可能存在臭虫，这将使它在某些情况下输出错误的代码。因此，认识和理解库将是非常重要的，这样才能够改正其中的错误。正因为如此，在本学时以前就没有提到 GUI 构造程序。但是，如果已经理解前面 23 个学时中的内容，你现在才具有使用 GUI 构造程序的权利。

24.5　问题与答案

问：如果我理解正确的话，需要 perl 才能编译 QtEz。我没有 perl，从哪里能够得到它？

答：perl 分发程序很可能包含在你的 UNIX/Linux 分发程序中。如果没有，可以从 www.perl.com 下载。

问：当我运行 QtEz 时，它报告 QTEZ 变量没有设置。但是我确实已经设置了该变量，这是为什么？

答：如果你在 X 终端仿真程序或虚拟控制台下设置 QTEZ 变量，之后又在另一个控制台下启动 QtEz，QtEz 将找不到该变量。为了为所有的 X 终端仿真程序或虚拟控制台设置 QTEZ 变量，你需要在启动脚本（如/etc/profile）中设置该变量。

问：我正在编译 QtEz（或 QtArchiture），它已经花费了好几个小时。是不是肯定出了问题？

答：不，如果计算机正在编译，就没有出现问题。QtEz 和 QtArchiture 都是非常大的应用程序，因此，编译将花费一定的时间。如果你的计算机是几年前的老机器，这一过程很可能需要几个小时。

问：我不知道应该选择哪个 GUI 构造程序。你认为呢？

答：如果你不反对使用 QtEz 界面的话，它是当前最好的选择。其功能最多，并且能够处理 GUI 实际构造过程中的许多工作。当然，如果在将来开发出其他 GUI 构造程序，情况也许会发生变化。

24.6 作　业

完成下面的问题和练习将有助于你牢记这一学时所学的 GUI 构造程序知识。

24.6.1 测验

1. 什么是 GUI 构造程序？
2. 是否有多个 Qt GUI 构造程序？
3. GUI 构造程序不能够帮助你做哪些工作？
4. 即使使用 GUI 构造程序，为什么你还必须了解 Qt 的工作机制？

24.6.2 练习

1. 启动你所喜欢的 GUI 构造程序。创建一个新项目和一个窗口（它与 GUI 构造程序中的相同）。为一个简单的文本编辑器创建 GUI，它具有一个菜单栏（有 File、Edit 和 Help 菜单）、一个状态栏和一个工具栏。在其中部，应该是一个大的 QmultiLineEdit 对象。将这个项目代码导出到源文件中，并编译和运行它。

附录 A　测验题答案

第 1 学时

1. 什么是 Qt?

Qt 是用于创建 GUI 程序的 C++类库。

2. 与其他同类产品相比，Qt 有哪些优点?

Qt 具有快速、可移植性和容易使用等优点。

3. 什么是 QPushButton 类?

QPushButton 类提供可点击按钮。它可以具有文本或位图标签。

4. QWidget 类有什么用途?

在 Qt 应用程序中，QWidget 用做工作空间。它是底层窗口，在其上可以放置其他对象，如按钮等。

5. a.setMainWidget(&mainwindow); 语句的意义是什么?

这段代码告诉 Qt，说明 mainwindow 对象是程序的主部件。当主部件关闭时，整个程序随之结束。

6. 从哪里查找 Qt Reference Document?

Qt Reference Document 包含在 Qt 分发程序的 doc 子目录中。

第 2 学时

1. OOP 代表什么意思?

OOP 代表 object-oriented programming，即面向对象程序设计。

2. 什么是类?

类是对一种特殊数据类型的描述。可以描述类由哪些数据和哪些函数组成。

3. 什么是对象?

如果你编写一个描述汽车的类，该类的一个对象就是某一辆汽车。可以为每个汽车创建一个对象。但是，对象也可以是一座房子、一条狗、或者是 GUI 程序中的一个图形按钮。

4. 什么是方法?

方法与成员函数相同，这个函数属于某个类。

5. 什么是类继承?

类继承就是基于已经存在的类构造新类的一种方法。可以使用经过测试的代码，只需添加你所需的功能。通过使用类继承，不必重复编写代码。

6. 使用 Qt 时为什么需要 OOP 知识?

Qt 是一个 C++类库。它是一个 OOP 库，使用 C++中所包含的 OOP 功能。因此，使用 Qt 时需要具有 OOP 知识。

第 3 学时

1. setMaximumSize()函数的作用是什么?

setMaximumSize()函数设置所讨论部件的最大尺寸。

2. setMinimumSize()函数的作用是什么?

setMinimumSize ()函数设置部件可以具有的最小尺寸。

3. setGeometry()函数的作用是什么?

setGeometry()函数用于设置部件的大小和位置。如果部件是一个窗口，之后它可以被移动或重新调整大小。

4. 在源程序中包含 qfont.h 头文件有什么作用?

使用 QFont 类能够格式化文本。

5. MyMainWindow w; 程序行的作用是什么?

它创建一个 MyMainWindow 对象，并执行类构造函数。

6. 为什么不必调用每个对象的 show ()函数?

当调用父部件的 show()函数时，子部件被自动显示。

7. 为什么输入 this 指针代替父部件?

this 指针表示在当前类中还未创建的对象。因此，如果设置 this 指针作为父部件，父部件将是在后面的 main()函数中所要创建的对象。

8. 什么是 qApp?

qApp 的创建与 this 指针想法相同。它是一个指向还未创建的 QApplication 对象（它在 main()中创建）的指针。

9. 为什么在 main ()函数中需要调用 a.exec ()函数?

在这一行，将程序控制权传递给 Qt 库。从这里开始，Qt 负责处理用户交互和其他程序操作。

第 4 学时

1. 什么是槽?

槽是一种特殊类型的成员函数，它可以被连接到信号。当发射信号时，槽（函数）将被执行。

2. 什么是信号?

信号是一种特殊类型的函数，它可以被连接到槽。当某个事件发生时，信号通知槽，之后，槽被执行。

3. 怎样将信号连接到槽?

使用 QObject::connect()函数。

4. 能否将多个槽连接到一个信号?

能够，这是可能的。只需为每个连接调用一次 connect()函数。

5. 什么时候能够调用 connect()函数而不指定定义它的类?

当从派生于 QObject 或其子类的类成员函数中调用时。

6. 能够将被连接的槽和信号断开吗?

是的，这是可以的。只需使用 QObject::disconnect()函数。

7. 在调用 connect()函数时，省略槽所属对象名称意味着什么?

这意味着槽在当前类中定义（也就是说，类是当前所定义类）。因此，只能从具有槽的类成员函数中执行这类调用。

8. 是否能够将一个信号连接到其他信号? 如果能，应该怎样操作?

可以。只需像通常一样使用 connect()函数，如：

```
connect( button, SINGAL( clicked( ) ), this, SINGAL( anothersignal( ) ) );
```

第 5 学时

1. QScrollView 有什么用途?

QScrollView 用于创建具有滚动条的窗口。

2. 哪个成员函数用于向 QScrollView 类添加对象?

QScrollView::addChild()函数用于向 QScrollView 类添加对象。

3. 按需滚动是什么意思?

按需滚动意味着只有当需要时才显示出滚动条（也就是说，当所有部件能够完全显示出来时不显示滚动条）。

4. 什么是 QMenuBar 和 QPopupMenu?

QMenuBar 和 QPopupMenu 是用于创建下拉菜单的 Qt 类。QMenuBar 对象表示整个菜单栏，QPopupMenu 对象代表单个菜单。

5. 什么时候需要调用 QMenuBar::insertItem ()?

当向菜单栏添加菜单时需要调用 QMenuBar::insertItem ()。

6. 什么是 QToolBar 和 QToolButton?

QToolBar 和 QToolButton 是用于创建工具栏的 Qt 类。QToolBar 表示实际工具栏，QToolButton 代表工具栏中的一个按钮。

7. QMainWindow 类适用于做什么?

QMainWindow 用于创建具有标准外观的应用程序。

8. 什么是 QMainWindow 对象中的中心部件?

中心部件是一个部件，QMainWindow 将向它周围添加菜单栏、工具栏和状态栏。例如，

在一个文本编辑器中，中心部件就是输入文本的空白区域。

9. 向基于 QMainWindow 的类添加工具栏时是否需要调用特殊的函数?

不需要。当创建工具栏时，QMainWindow 能够处理它。

10. 什么是 QPixmap?

QPixmap 是一个管理位图文件类。其怎样使用的详细信息请参看程序清单 5-5。

第 6 学时

1. 不同类型的按钮都有什么作用?

按钮（QPushButton）常用于使程序显式执行一些操作。单选按钮用于从多个选项中选择一种。复选按钮从多个选项中做出多个选择。

2. 检查是否点击 QPushButton 对象使用什么信号?

clicked()信号。

3. 布置按钮需要使用哪个类?

QButtonGroup 类。

4. 哪个函数能够设置 QLabel 对象中文本的对齐方式?

setAlignment()函数。这个函数也用于设置很多 Qt 部件的对齐方式。

5. 怎样改变 QLabel 对象中文本的大小、字体和样式?

调用 QPushButton::setFont()。这个函数用 QFont 对象做参数。这里是一个例子：

```
bl->setFont( QFont( "Times", 16, QFont::Bold ) );
```

6. 当基于 QTableView 创建子类时，除构造函数外，还有哪个函数需要实现?

paintCell()函数。

7. 为什么需要实现这个函数?

如果不以有效的方式实现这个函数，QTableView 不知道怎样绘制表元。

8. 使用哪个类创建表头?

QHeader 类。

第 7 学时

1. 什么是选择部件?

与其名称含义相同，选择部件是用于让用户选择文本、位图和数值的部件。

2. Qt 提供哪两种类型的文本选择部件?

列表框（QListBox）和组合框（QComboBox）。二者非常类似，但组合框所占用的空间较少。

3. QSplitter 类的用途是什么?

当你想使用户能够改变两个或多个部件大小时，使用 QSplitter 类。

4. QWidgetStack 类的用途是什么?

可以向 QWidgetStack 添加部件，之后能够控制当前显示哪个部件。

5. 当使用部件栈时，什么时候应该使用整数标识号代替部件指针？

在编译时，当不能完全确定需要使用哪个部件时，这一功能非常有用。程序中的事件将判断将哪个部件插入到部件栈中。这时，需要使用整数标识号。

6. 尽管没有明确指出这一学时中的选择部件，但你学习了两种用于数值的选择部件，你能指出它们的名字吗？

滑动框部件（QSlider）和微调框部件（QSpinBox）。

7. 与微调框相比，滑动框的优点是什么？

当使用滑动框而不是微调框时，能够很容易地向用户传送可能的数值。相反，微调框占用的空间较少。

第 8 学时

1. QLineEdit 和 QMultiLineEdit 间的区别是什么？

QLineEdit 用于从用户读取短文本信息，如口令等。QMultiLineEdit 用于向用户显示大量的文本信息或者让用户输入大量的文本。

2. 向用户显示文本文件时应该使用哪个类？

很可能是 QMultiLineEdit。

3. 如果只考虑成员函数，那么，QLineEdit 和 QMultiLineEdit 间最本质的区别是什么？

QMultiLineEdit 包含的函数能够使程序在准确位置插入文本。

4. 如果不使用 insertItem()函数向列表视图中插入条目，那应该使用哪个方法？

父-子方法。

5. 当双击列表视图中的条目时将发射哪个信号？

doubleClicked()信号。

6. 当在列表视图中按回车键时将发射哪个信号？

returnPressed()信号。

7. 什么是进程条？

进程条是一个智能部件，它向用户显示完成一个或多个任务还需要多长时间。

8. 在应用程序中，什么时候应该使用进程条？

当应用程序执行非常耗时的任务时。

第 9 学时

1. 什么类用于创建图形？

QPainter 类用于创建图形。

2. setPen()函数的功能是什么？

使用 setPen()函数能够改变你所使用的画笔类型。

3. 画刷有什么用途？

画刷用于填充一个图形内部。

4．是否有多种填充模式供选择使用？

是的，有多种填充模式。所有填充模式的描述如表 9-1 所示。

5．为什么要使用 QPainter 类显示文本？

是的，为什么呢？如果你的目标只是显示文本，最好使用 QLabel。

6．什么是 RGB？

RGB 代表 Red/Green/Blue（红/绿/蓝），它是一种颜色定义方法。

7．什么是调色板？

一个调色板包含 3 个颜色组，它们构成一个完整的颜色集合——每一个代表一种状态（活动程序、禁止程序和正常程序）。

8．在纸上打印出一个圆需要做哪些工作？

只需像通常一样向 QPainter 对象绘制一个圆。

第 10 学时

1．什么是对话框？

对话框是一个以某种方式与用户交互的部件（例如，询问用户是否需要继续等）。

2．Qt 是否提供文件选择对话框？

提供，使用 QFileDialog 类能够很容易地创建文件对话框。

3．是否能够选择多个文件？

可以，只要使用 QFileDialog::getOpenFiles()函数（注意结尾的 s）。

4．为什么你想让用户选择颜色？

如果在创建一个绘图程序，用户很可能需要选择使用不同的颜色。

5．什么类用于创建字体对话框？

QFontDialog。

6．如果你想向用户提问一个简单的问题，应该使用哪个对话框？

最简单的方法是使用 QMessageBox 对话框。但是，你也可以为此创建一个用户定义对话框。

7．如果你准备打开 50 个文件，你应该设置的最大进度值为多少？

当然是 50。之后，每当打开一个文件时调用一次 QProgressDialog::setProgress()函数。

8．能使用 QDialog 基类创建自己的选项卡对话框吗？

可以，但不应该这样做。尽管使用 QDialog 这样做是可能的，但是你应该使用 QTabDialog。Qt 开发者已经为你做了大量的工作。

第 11 学时

1．什么是布局管理器？

布局管理器是一个用于布置部件的 Qt 类。

2．布局管理器能帮助你做什么？

使用布局管理器，你不必手工定义部件的位置-坐标，并能节省许多工作。

3．QVBoxLayout 怎样组织子部件？

QVBoxLayout 按列方式组织其子部件。

4．QHBoxLayout 怎样组织子部件？

QHBoxLayout 按行方式组织其子部件。

5．QGridLayout 怎样组织子部件？

QGridLayout 以网格方式组织其子部件。

6．在 QGridLayout 布局管理器中哪两个数代表左上角的部件？

0 和 0。

7．什么时候需要使用嵌套布局管理器？

为了获得更原始和更复杂的布局。

8．什么时候必须调用 QLayout::Avtivate()函数？

在旧的 Qt 1.4x 版本中，必须调用这个函数激活布局管理器。

第 12 学时

1．在 Qt 中哪个类代表一个文件？

QFile 类。

2．QTextStream 类的用途是什么？

它用于在文件和程序之间创建流。之后，这个流能够用于从文件向程序传递信息。

3．QDir 类做什么用？

它读取目录，并为每个目录项创建一个向量（记住，在向量中，使用[]操作符能够访问其每一个元素）。

4．怎样读取一个目录中的第 5 个项目？

调用 myDirObject[4]。记住，4 代表第 5 个参数，因为起始号为 0。

5．什么时候需要使用 QFileInfo 类？

当需要文件信息时（例如文件长度等）。

第 13 学时

1．为什么你认为 QValidator 是一个抽象类？

只是因为不可能预测程序员想使用的所有不同的验证方法。

2．是否有预定义验证类？

有，QDoubleValidator 和 QIntValidator。

3．如果你在前一个问题中回答是，那么你能告诉这一验证类（或这些验证类）的用途是什么吗？

用于验证浮点数和整数。

4．哪个 Qt 类能够使用验证类？

QLineEdit、QSpinBox 和 QComboBox。

5．你需要调用哪个函数来告诉一个对象使用验证？

setValidator()函数。

6．当创建你自己的验证类时，实际验证代码插入在哪个虚函数内？

validate()函数。

7．你可用某个虚函数纠正基本可接受的字符串，这个函数是什么？

fixup()函数。

第 14 学时

1．什么时候应该使用 QStack 容器类？

当你想创建一个元素列表并用反序检索它们时。

2．什么时候应该使用 QQueue 容器类？

当你想创建一个元素列表并用与它们插入时相同的顺序检索它们时。

3．什么是散列表？

一个散列表是一个元素列表，其中的每个元素能够通过一个字符串键值或整数键值访问。

4．如果想创建具有整数键值而不是字符串键值的散列表，应该使用哪个类？

QIntDict 类。

5．QCache 有什么用途？

用于创建长度能够控制的散列表。

6．当一个缓存到达其最大总成本值时将发生什么问题？

最少使用的元素将被删除。

7．什么时候需要使用迭代？

当遍历一个目录类（QDict、DIntDict、QCache 或 QIntCache）时。

第 15 学时

1．在显示 GIF 动画之前你需要做的第一件事是什么？

需要编译的 Qt 库支持 GIF。

2．有什么方法能够加速动画吗？

有，可以使用 QMovie::setSpeed()函数设置动画速度。

3．有什么方法能够读取动画的当前速度吗？

有，通过调用 QMovie::speed()函数。

4．Qt 是否有它自己的图像格式？

有，用 QPicture 类能够保存和装载这种格式的图像（通常，Qt 图像格式用.pic 文件扩展名存储）。

5. 为什么需要使用 Qt 图像格式?

因为它是一种与平台无关的图像格式，所有 Qt 平台都支持。

6. 在 Qt 中可以使用 JPG 图像吗?

可以，如果你的系统中已经安装了 JPG 库。如果不能完全确定程序的所有用户都使用 JPG 库，则不要使用 JPG 图像。

第 16 学时

1. 怎样做才能使用标准组合键实现文本的拷贝和移动操作?

不需做任何额外事情。

2. 用标准 Qt 剪贴板函数能够实现图像的剪切和粘贴吗?

可以，通过使用 QClipboard::setImage()和 QClipboard::image()函数。

3. 如果想使用剪贴板，应该基于哪个类?

QMimeSource()类。

4. 只使一个部件的某个区域能够接收拖放对象，这可能吗?

可以，通过使用 mouseMoveEvent()函数就可以实现。

5. 当将对象放置在一个能够接收对象的区域时，应该调用哪个函数?

dropEvent()函数。

6. 使一些部件只能从中拖动对象而不能在其上放置对象，或者使另一些部件只能放置对象而不能从中拖动对象。这可能吗?

可以，只要不为该部件定义拖、放功能。

第 17 学时

1. 什么是 KDE?

KDE 是一套 GUI 程序，它们构成一个全功能的桌面环境。

2. 当编写 KDE 程序时，应该使用哪个类代替 QApplication?

KApplication 类。

3. 什么是 kapp?

kapp 与常规 Qt 程序中的 qApp 的功能相同。它是一个指向还未创建的 KApplication 对象的指针。

4. 在 KDE 程序中应该使用哪个类创建主窗口?

KTMainWindow 类。

5. KApplication::getHelpMenu()有什么用途?

创建符合 KDE 标准 Help 菜单。

6. 在 KDE 程序中是否有显示 HTML 文档的好方法?

有，它非常简单，如果你使用 KHTMLWidget 类。

第 18 学时

1．KDE 核心库中有哪些内容？

它包含一些最低级类，如 KApplication 和 KProcess。

2．如果你想在程序中启动一个子进程，应该使用哪个类？

KProcess 类。

3．如果子进程正好是一个外壳命令，你应该使用哪个类？

应该使用 KShellProcess 类。

4．与 QPixmap 相比，KPixmap 具有哪些优点？

KPixmap 包含两种新的颜色模式，它们使程序在 256 种颜色下显示时效果变得更好。

5．在 KDE 程序中能够将一个窗口最小化吗？

可以，用 KWM 类即可实现。实际上，通过这个类你能够管理所有窗口管理器操作。

6．libkdeui 中包含哪种类？

libkdeui 具有用户接口类。这些类用于创建实际用户界面。

7．当编写 KDE 应用程序时，是否总需要使用 libkdeui 中的类构造用户界面？

绝对不需要。你可以使用 Qt 类来实现，并仍能够保持程序与 KDE 兼容。

8．当需要执行一些文件 I/O 操作时，你是否应该使用 libkfile 中的类？

不需要，目前在 libkfile 中还没有这样的类。

第 19 学时

1．什么是 OpebGL？

OpenGL 库用于创建高性能三维图形。

2．什么是 MESA？

MESA 是一个免费的 OpenGL 实现。

3．什么是 libqgl？

libqgl 是 Qt OpenGL 扩展名称。Qt OpenGL 类定义在这个库中。

4．什么时候应该使用 OpenGL？

当需要程序实现一些图像增强操作时。例如，绘图操作。但坚持使用 QPaint 是最安全的。

5．OpenGL 的弱点是什么？

OpenGL 最明显的弱点是缺乏用户交互功能。在 OpenGL 程序中难以实现按钮、菜单等。

6．QGLWidget 的用途是什么？

QGLWidget 类表示 OpenGL 部件。你应该将 OpenGL 代码添加到 QGLWidget 子类中。

第 20 学时

1．为什么应该使用 Qt 去开发 Netscape 插件？

因为你可以使用已经熟悉的 Qt 库，而不必学习怎样使用 Netscape Plugin SDK 所提供

API。

2．QNPlugin::getMIMEDescription()函数的功能是什么？

QNPlugin::getMIMEDescription()函数用于设置 MIME 类型、文件扩展名和 MIME 描述。

3．什么是 MIME 类型？

MIME 为文件或流数据指定一种特殊类型，它使浏览器知道怎样去处理它。

4．假设已经创建了一个显示 BMP 图像的插件。你打开一个具有很多 BMP 图像的文档。有多少个 QNPlugin 对象被创建？

只有一个。

5．在第 4 个问题中，创建有多少个 QNPInstance 对象？

为每个 BMP 图像创建 1 个。

6．QNPInstance::newWindow()函数的用途是什么？

浏览器使用 QNPInstance::newWindow()函数读取新的 QNPWidget 对象（实际为 QNPWidget 的一个子类）。

7．什么时候浏览器调用 QNPInstance::newStreamCreated()函数？

当浏览器向插件提供数据时调用 QNPInstance::newStreamCreated()函数。

8．在 QNPWidget 子类中能够找到哪一类代码？

常规 Qt 代码。

9．什么时候需要创建 QNPStream 子类？

当需要插件通过流接收数据时。

10．HTML 标记<EMBED>的用途是什么？

<EMBED>标记用于在 HTML 文档中嵌入插件。

第 21 学时

1．什么是用户空间文本？

用户空间文本是所有用户可见文本。

2．为什么应该对所有的用户空间文本使用 QString？

因为 Qstring 用 Unicode 编码，这能够保证文本以正确的方式显示。在某些情况下使用 QString 还能够提高性能。

3．什么时候需要使用 tr()函数？

当有字符串需要翻译时必须调用 tr()函数。

4．findtr 实用程序的用途是什么？

findtr 实用程序用于为一个或多个文件中的翻译创建一个数据库。它只是查找所有的 tr()函数调用，并为每个新字符串添加一项。

5．什么时候需要实用 msg2qm 实用程序？

经常，当你想使用翻译功能时。

6．什么时候需要实用 mergetr 实用程序？

当源文件被修改之后，如果你想将这些修改合并到相关语言的 PO 文件中，mergetr 就很有用。

7．哪个类在程序中执行实际翻译操作？

QTranslator 类。

8．QApplication::installTranslator()函数做什么？

QApplication::installTranslator()函数告诉 QApplication 对象有一个翻译器可以使用，并且应该使用它。

9．为什么需要调用 Q_OBJECT 宏，并对使用翻译的类使用元对象编译器？

因为翻译功能由 moc 实现。

10．QDate 用于处理什么？

QDate 用于显示当前日期，或用于处理日期值。

11．QTime 用于处理什么？

Qtime 用于显示当前时间，或用于处理时间值。

12．什么时候应该考虑使用 QDateTime 类？

当需要同时显示日期和时间时，应该使用 QDateTime 类。为此，你也可以使用两个独立的 QDate 和 QTime 对象，但是，用 QdateTime 类更方便。

第 22 学时

1．使用 Qt 可以编写网络应用程序吗？

可以，通过使用 Qt 的网络扩展。

2．什么是 POSIX？

POSIX 是一个操作系统标准。所有 POSIX 变体，包括 Windows NT，都支持这个标准。

3．什么是 POSIX.1？

POSIX.1 是 POSIX 中程序员最感兴趣的一部分。它在函数一级定义怎样访问系统相关的功能。

4．Qt 中是否有一些函数在 UNIX 和 Windows 下完全不兼容？

有几个，但是，其差别很小，通常也不会引起任何问题。

5．什么是 tmake？

tmake 是一个小巧的实用程序，它帮助你为 Qt 支持的所有平台创建编译文件。

6．什么是项目文件？

在项目文件中向 tmake 传递关于项目的某些指令——例如，包含哪些文件等。

7．什么是 progen？

progen 是一个实用程序，它帮助你创建项目文件。当处理大型项目时，progen 尤其有用。

第 23 学时

1．什么时候应该使用 qDebug()函数？

qDebug()函数用于通知某个程序事件（也就是当某个事件发生时）。

2．什么时候应该使用 qWarning()函数？

qWarning ()函数用于通知程序中发生的意外事件。

3．什么时候应该使用 qFatal()函数？

当程序中发生相当严重的事件时，使用 qFatal()函数，程序因此需要退出。

4．ASSERT()宏的作用是什么？

ASSERT()宏用于通知你一个布尔值不为 TRUE。

5．CHECK_PTR()宏的作用是什么？

CHECK_PTR()宏用于检查一个指针是否为空。这个宏适用于检查一个对象是否被正确创建。

6．什么是调试器？

调试器是一种程序，使用它能够获得程序内部信息。有很多调试器，你还能够准确控制执行程序中的哪一部分，以及检查变量状态。

7．怎样告诉 gdb 你想处理哪个文件？

将文件名作为 gdb 参数来告诉 gdb 应该处理哪个文件。

8．使用什么 gdb 命令能够设置断点？

使用 gdb 的 break 命令设置断点。

9．gdb 中 run 命令的作用是什么？

run 命令启动程序。

10．你猜想一下，哪个命令能够退出 gdb 调试器？

当然是 quit 命令。

第 24 学时

1．什么是 GUI 构造程序？

GUI 构造程序是一个以图形方式创建 GUI 界面的程序。

2．是否有多个 Qt GUI 构造程序？

是的，有几个。这一学时中介绍了 3 个。

3．GUI 构造程序不能够帮助你做哪些工作？

GUI 构造程序不能编写与 GUI 无关的功能。尽管一些 GUI 构造程序也包括一些其他功能，但是，GUI 构造程序的基本用途是构造 GUI。

4．即使使用 GUI 构造程序，为什么你还必须了解 Qt 的工作机制？

为了解决 GUI 构造程序不能自动处理的事情。此外，修改 GUI 构造程序可能产生的错误也需要这一知识。

附录 B　常用 Qt 类

Qt 由大量类组成，为了完全掌握该库，你应该了解 Qt 所带的这些类以及它所提供的内容。表 B-1 列出最常用的 Qt 类，并简短描述这些类。这个表可以用作 Qt 类参考。

表 B-1　　常用 Qt 类

类	描　述
QAccel	处理快捷键
QBoxLayout	布局管理器，用于在水平或垂直方向上布置部件
QButtonGroup	用于组织按钮
QCDEStyle	用于创建 CDE 风格的部件
QChar	存储 Unicode 字符
QCheckBox	用于创建具有标签的多选按钮
QClipboard	访问系统剪贴板
QColor	表示一种 RGB 颜色
QColorDialog	提供颜色选择对话框
QComboBox	提供组合框
QCString	提取传统的 C 字符数组 char *
QCursor	提供一个用户定义形状的鼠标指针
QDate	处理日期
QDateTime	处理日期和时间组合值
QDialog	用于创建用户对话框的基类
QDir	遍历或处理目录
QDoubleValidator	预定义验证类，用于验证浮点数
QDragEnterEvent	当一个部件被拖到一个部件上时所发射的信号
QDragLeaveEvent	当一个拖放对象离开一个部件上时所发射的信号
QDragMoveEvent	在拖放期间所发射的信号
QDragObject	代表一个拖放事件中被拖动的对象
QDropObject	当一个拖放对象被放置到一个部件上时所发射的事件
QFile	处理文件
QFileDialog	提供文件选择对话框
QFont	表示一种能够用于绘制文本的字体
QFontDialog	提供一个字体选择对话框

续表

类	描　述
QFontInfo	提供一种字体的通用信息
QGLContext	封装 OpenGL 绘图环境
QGLFormat	指定 OpenGL 显示环境的显示格式
QGridLayout	以网格形式布置部件
QGroupBox	提供一个带标题的分组框
QHBoxLayout	在水平方向上布置部件
QHButtonGroup	组织水平方向上的按钮
QHeader	提供表标题
QImage	表示一个可以处理的图像
QImageDrag	提供一个图像型拖放对象
QImageIO	提供图像存储和装载操作函数
QKeyEvent	表示一个键盘事件
QLabel	用于显示静态文本
QLCDNumber	表示以 LCD 状所显示的数字
QLineEdit	单行文本输入域
QListView	提供一个可选择元素列表
QListViewItem	表示列表视图中的一个元素
QMainWindow	为应用程序提供一个标准主窗口
QMenuBar	提供一个水平菜单栏
QMessageBox	用于向用户显示简短文本信息
QMotifStyle	以 Motif 风格显示部件
QMouseEvent	描述一个鼠标事件
QMoveEvent	描述一个移动事件
QMovie	用于显示动画
QMultiLineEdit	提供一个文本显示和输入多行区域
QNPInstance	代表一个 Netscape 插件实例
QNPlugin	Netscape 插件核心内容
QNPStream	为 QNPInstance 提供一个数据流
QNPWidget	表示一个插件窗口
QPainter	用于执行各种绘图功能
QPallete	为每个部件调色板提供一个颜色组
QPen	定义 QPainter 所绘制的线型
QPicture	以 Qt 图像格式保存图像
QPixmap	代表一个位图，例如，用于显示在屏幕上
QPlatinumStyle	用于以 Platinum 样式显示部件
QPNGImagePacker	用于创建 PNG 动画
QPoint	表示平面中的一个点
QPointArray	提供一个点数组

续表

类	描 述
QPopupMenu	提供一个弹出菜单
QPrinter	用于打印
QProgressBar	提供一个进度条
QProgressDialog	提供一个进度对话框
QPushButton	提供一个具有标签的可点击按钮
QRadioButton	提供一个具有文本标签的单选按钮
QRect	代表平面中的一个矩形
QRegExp	表示一个常规表达式
QScrollView	提供一个按需水平和垂直滚动条
QSlider	提供一个水平或垂直滑动框
QSpinBox	提供一个通过点击两个箭头即可选择数字的部件
QSpitter	提供一个控制其他部件大小的部件
QStatusBar	提供一个状态栏
QString	用于存储 Unicode 字符串
QStringList	提供一个字符串列表
QStyle	封装一个 GUI 外观
QTabDialog	提供一个选项卡对话框
QTableView	表示表的抽象类
QTextBrowser	提供一个格式化文本浏览器
QTextCodec	提供文本编码转换操作
QTextDrag	代表文本拖放操作
QTextStream	一个类，它使用 QIODevice 中的文本
QTextView	格式化文本视图
QTime	表示时间类
QTimer	用于测量时间
QToolBar	提供一个工具栏
QToolButton	表示工具栏中的一个按钮
QToolTip	为一个部件提供提示
QTranslator	实现程序翻译功能
QUriDrag	提供一个 URI 拖放对象
QValidator	提供输入文本验证
QVBoxLayout	在垂直方向上布置部件
QWidget	所有用户接口对象的基类
QWizard	向导类
QXtApplication	允许 Qt 和 Xt/Motif 部件混合
QXtWidget	与上一个相同